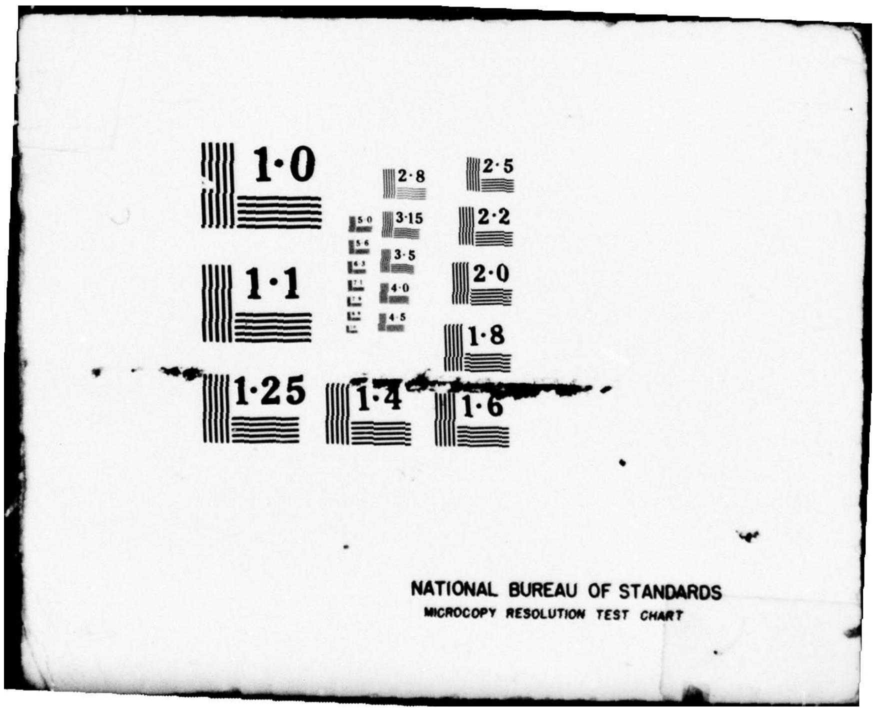
AD-A068 112 MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTE--ETC F/G 9/2
A SEMANTIC DATA BASE MODEL AND ITS ASSOCIATED STRUCTURED USER I--ETC(U)
AUG 78 D MCLEOD N00014-76-C-0944
UNCLASSIFIED MIT/LCS/TR-214 NL

1 OF 4
ADA
068112


1·0
2·8
2·5
5·0
5·6
3·15
2·2
1·1
3·5
2·0
4·0
4·5
1·8
1·25
1·4
1·6
NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

MIT/LCS/TR-214

A SEMANTIC DATA BASE MODEL AND ITS ASSOCIATED STRUCTURED USER INTERFACE

Dennis McLeod

This research was supported by the
Advanced Research Projects Agency
of the Department of Defense
and was monitored by the Office of Naval Research
under Contract No. N00014-76-C-0944.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (14) MIT/LCS/TR-214	2. GOVT ACCESSION NO. ✓	3. RECIPIENT'S CATALOG NUMBER (9) Doctoral thesis,
4. TITLE (and Subtitle) (6) A Semantic Data Base Model and Its Associated Structured User Interface,		5. TYPE OF REPORT & PERIOD COVERED Ph.D. Thesis, August 1978
		6. PERFORMING ORG. REPORT NUMBER MIT/LCS/TR-214
7. AUTHOR(s) (10) Dennis/McLeod		8. CONTRACT OR GRANT NUMBER(s) (15) N00014-76-C-0944
9. PERFORMING ORGANIZATION NAME AND ADDRESS MIT/Laboratory for Computer Science 545 Technology Square Cambridge, MA 02139		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS ARPA - Department of Defense 1400 Wilson Boulevard Arlington, VA 22209		12. REPORT DATE (11) August 1978
		13. NUMBER OF PAGES 381
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) ONR - Department of the Navy Information Systems Program Arlington, VA 22217 (12) 379p.		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release, distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

data base management
data models
data semantics
data base modelling
data definition
data base design

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The conventional approaches to the structuring of data bases provided in contemporary data base management systems are in many ways unsatisfactory for modelling data base application environments. The features they provide are too low-level, computer-oriented, and representational to allow the semantics of a data base to be directly expressed in its structure. The *semantic data model (SDM)* has been designed as a natural application modelling mechanism that can capture and express the structure of an application environment. The features of the SDM correspond to the principal intensional structures → over

20. cont

naturally occurring in contemporary data base applications. The SDM provides a rich but limited vocabulary of data structure types and primitive operations, striking a balance between semantic expressibility and the control of complexity. Furthermore, facilities for expressing derived (conceptually redundant) information are an essential part of the SDM; derived information is as prominent in the description of an SDM data base as is primitive data.

The SDM is designed to enhance the effectiveness and usability of data base systems:

1. SDM data bases are to a large extent self-documenting, in the sense that the description and structure of a data base are expressed in terms which are close to those used by users in describing the application environment.
2. The SDM can support powerful user interface facilities, and can improve the user interface effectiveness for a variety of types of users (with varying needs and abilities). Significantly, SDM data bases capture information in a form useful to its users, and allow derived information helpful in new data base uses to be defined in the data base structure. In particular, the SDM supports an incremental, interactive interface for the "naive" nonprogrammer (an *interaction formulation advisor*), which guides the user through the data base and the process of formulating a query or update request.
3. The SDM can be used as a tool in the data base design process. The SDM aids in the identification of relevant information in a data base application environment, as well as in organizing that information and relating it to its possible uses. This can greatly improve the design of lower-level, conventional data bases.

The use of the SDM is not dependent on the successful implementation of a new data base management system that directly supports it. There are many data base management systems in use today which represent a considerable investment on the parts of their developers and users; the SDM can be effectively used in conjunction with these existing data base systems to enhance their effectiveness and usability. For example, a prototype interaction formulation advisor demonstrates that the SDM can be used as a user-oriented "front end" to a conventional data base system; an analysis of application modelling with the SDM illustrates its effectiveness in improving and simplifying the data base design process.

MIT/LCS/TR-214

A SEMANTIC DATA BASE MODEL AND ITS ASSOCIATED STRUCTURED USER INTERFACE

by

DENNIS MCLEOD

August, 1978

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under Contract No. N00014-76-C-0944.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LABORATORY FOR COMPUTER SCIENCE

CAMBRIDGE MASSACHUSETTS 02139

ABSTRACT

The conventional approaches to the structuring of data bases provided in contemporary data base management systems are in many ways unsatisfactory for modelling data base application environments. The features they provide are too low-level, computer-oriented, and representational to allow the semantics of a data base to be directly expressed in its structure. The *semantic data model (SDM)* has been designed as a natural application modelling mechanism that can capture and express the structure of an application environment. The features of the SDM correspond to the principal intensional structures naturally occurring in contemporary data base applications. The SDM provides a rich but limited vocabulary of data structure types and primitive operations, striking a balance between semantic expressibility and the control of complexity. Furthermore, facilities for expressing derived (conceptually redundant) information are an essential part of the SDM; derived information is as prominent in the description of an SDM data base as is primitive data.

The SDM is designed to enhance the effectiveness and usability of data base systems:

1. SDM data bases are to a large extent self-documenting, in the sense that the description and structure of a data base are expressed in terms which are close to those used by users in describing the application environment.
2. The SDM can support powerful user interface facilities, and can improve the user interface effectiveness for a variety of types of users (with varying needs and abilities). Significantly, SDM data bases capture information in a form useful to its users, and allow derived information helpful in new data base uses to be defined in the data base structure. In particular, the SDM supports an incremental, interactive interface for the "naive" nonprogrammer (an *interaction formulation advisor*), which guides the user through the data base and the process of formulating a query or update request.
3. The SDM can be used as a tool in the data base design process. The SDM aids in the identification of relevant information in a data base application environment, as well as in organizing that information and relating it to its possible uses. This can greatly improve the design of lower-level, conventional data bases.

The use of the SDM is not dependent on the successful implementation of a new data base management system that directly supports it. There are many data base management systems in use today which represent a considerable investment on the parts of their developers and users; the SDM can be effectively used in conjunction with these existing data base systems to enhance their effectiveness and usability. For example, a prototype interaction formulation advisor demonstrates that the SDM can be used as a user-oriented "front end" to a conventional data base system; an analysis of application modelling with the SDM illustrates its effectiveness in improving and simplifying the data base design process.

ACKNOWLEDGEMENTS

I am deeply indebted to a large number of people who have helped make this thesis possible. Here, I can only hope to mention those who have been most directly involved in the thesis itself:

My thesis supervisor, Michael Hammer, has provided nearly four years of guidance, advice, technical expertise, financial support, and encouragement. His continued confidence has enabled me to make this thesis a reality. I have the utmost respect for him, and owe him a great deal.

My readers, J. C. R. Licklider and Michael Zisman, have helped greatly in making this document more complete and intelligible.

Joseph Weizenbaum helped get me started in a Ph.D. program at MIT, and gave me the opportunity to gain valuable experience in teaching and presenting technical material.

My good freinds and "data base" colleagues at the MIT Laboratory for Computer Science provided many hours of helpful discussions, and shared with me enumerable old jokes. Particularly, I would like to thank Edward Cardoza, Dan Carnese (whose ability to make constructive comments on my work never ceases to astound me), Arvola Chan, Jay Kunin, Bahram Niamir, Sunil Sarin, and Stan Zdonik (who helped my wrestle with the "philosophical" underpinings of the SDM).

The system "hackers" in the Programming Technology Division at MIT provided me with endless good software and advice, and exhibited incredible patience with me as a "naive" user of sorts. I owe special thanks to Tim Anderson, Russ Atkinson, Bruce Daniels, and Dave Lebling.

The CLU group, headed by Barbara Liskov, provided me with a tremendously useful programming language and environment in which to work. The members of this group who answered my many questions deserve special thanks: Russ Atkinson, Eliot Moss, Bob Scheifler, and Alan Snyder.

My data base colleagues outside MIT have helped greatly in commenting on the research documented in this thesis, particularly: Phil Bernstein, Don Chamberlin, Ted Codd, Pat Griffiths, John Mylopoulos, Arie Shoshani, and John Smith.

My institutions, MIT and the Laboratory for Computer Science, gave me a valuable education, and the resources necessary to conduct research. The research described in this thesis was supported, in part, by the Advanced Research Projects Agency of the

Department of Defense, and was monitored by the Office of Naval Research under contract number N00014-76-C-0944.

I must acknowledge the now demised vessel (rust bucket) *Argo Merchant* (and its former captain), whose sad states spawned my interest in coastal and tanker monitoring, and gave rise to the principal example data base used in this thesis.

My parents have guided me to the point where this thesis was possible, and have provided me with their constant support and love. I can never thank them enough.

Mary Rykowski has given me the moral support, understanding, and love that enabled me to survive a Ph.D. program. She has put up with my endless ramblings, and has made life enjoyable despite my long hours these past few months. I dedicate this thesis to her.

NOTE

For brevity, throughout this thesis, we use "he", "him", and similar pronouns where we mean "he/she", "him/her", etc.

Key words: data base management, data models, data semantics, data base modelling, data definition, data base design.

This report is a minor revision of a thesis of the same title submitted to the Department of Electrical Engineering and Computer Science in August 1978 in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

TABLE OF CONTENTS

1. INTRODUCTION

Many computerized information systems are heavily oriented to the use and management of data; such systems accomodate the storage and maintenance of large amounts of information, and provide access to it. The computerized data banks associated with these information systems are increasingly numerous and essential in modern society; ready and convenient access to the data in these data banks is demanded by nearly all managerial, administrative, and decision making functions, in both the public and private sectors. Typical applications in which such repositories of data are critical include accounting systems, parts inventory, airline reservations, banking records, automobile registration, insurance policy management, and environmental standards monitoring. Indeed, both government and private industry are investing more and more time, effort, and money in the construction and maintenance of their data bases.

The term *data base*, in its most general interpretation, means a collection of information. However, we adopt a more restricted definition of the term here: a data base is a collection of "formatted" information. This means that the information is highly structured, in that the size of the data base is significantly larger than the size of its description. Such formatted data bases are classically organized into *files* of *records*. A file is intended to correspond to some logical unit of information in the application. Each record in a file has the same basic format: each record has the same *fields*, but may have different values in those fields. For example, a payroll data base might contain a file for employees, with fields "Name", "Employee Number", "Address", and so forth.

The contents of a computerized data base are intended to represent a snapshot of the state of an application system, and the changes to the data base over time reflect the sequence of events occurring in the application environment. In other words, a data base is a *model* of a real world system. Therefore, it is appropriate that the structure of the data base mirror the structure of the system that it models. A data base whose organization is based on naturally occurring structures should be easier for a data base designer to construct and modify than one that forces him to translate the primitives of his problem domain into artificial specification constructs. Similarly, a data base user should find it easier to understand and use a data base if it can be described to him in terms of concepts with which he is already familiar.

In this thesis, we present a new data base structuring method which allows the organization of a data base to reflect the semantics of the application environment it is intended to model. The specific emphasis here is on the effectiveness of this mechanism in supporting improved data base understandability and accessibility.

In the most general sense, our work is intended to make a contribution to the development of people-centered computing systems. Specifically, we are attempting to make some progress in providing data base users with a data base view and user interface which is easy and natural for them to understand and utilize. The purpose of this thesis, then, is to give meaning to these general goals, and to describe our approach to making progress towards them.

1.1. The Data Base Environment

In many information systems which rely heavily on data bases, the facilities for storing, managing, and providing access to data are embedded in the software system which supports the application. The principal responsibility for organizing and maintaining a data base managed in this way is assigned to the programmers who wish to make use of it. For example, many data base applications rely solely on the data management (organization, storage, and accessing) facilities of a conventional computer file system.

The principal problem associated with this approach is the heavy burden placed on the applications programmer in designing and maintaining a data base. Typically, the file system he uses has very limited capabilities to perform important data management tasks, such as supporting multiple access keys into data base records, providing the ability to select relevant data items by an expression involving the fields of a record, and so forth. As a consequence, the programmer has to build and maintain his own data management tools. This implies that he must be concerned with low level detail which is not relevant to his principal tasks: he must be aware of the details of the physical structure and representation of a data base, and he must write programs that explicitly manipulate these physical structures.

It is well known that serious problems of programmer productivity and software reliability result from the failure to separate the behavior of an "abstraction" from its implementation. Data bases are no exception to this rule. Software developed specifically for and tailored to a particular data base application tends to be designed and constructed in an ad hoc fashion;, this results in inefficiencies and the repeated rediscovery and implementation of the same basic functions. These software systems often exhibit poor modularity and are difficult to understand and modify. Finally, the data in many of these systems is repeatedly stored; each specific application may have its own data base. This results in the problems associated with maintaining the consistency of multiple copies of data and of effectively sharing information among the various tasks in the application.

One of the most important ramifications of this "do it yourself" approach to data organization is its adverse impact on user accessibility. Requests involving the retrieval of specific information from a data base must be funnelled through programmers: a time-consuming and costly process. This often encourages the end-user to view a data base as a very rigid and inflexible tool. As a consequence, the information in a data base is often greatly underutilized.

1.2. Data Base Management Systems

In response to the problems associated with embedding data management facilities in applications software, the concept of a *data base management system* was developed. A data base management system (DBMS) is intended to take on more of the responsibility for the storage and control of data bases than conventional file systems. For example, a DBMS may have the following kinds of capabilities:

1. support the organization of data into cohesive units, and allocate and maintain the storage for the data,

2. provide a set of primitives and/or a language for accessing the data, and for making changes to a data base,

3. accomodate physical integrity control, to allow a data base to be restored after a hardware, software, or communications failure,

4. control the access of concurrent data base users, so that they do not interfere with one another,

5. limit the access of particular users to certain information in a data base, thereby allowing data protection restrictions to be imposed,

6. control the semantic integrity of the data, by attempting to prevent the entry of clearly meaningless or erroneous information into a data base.

Most commercial and research prototype data base management systems are largely application environment independent, in that they are capable for supporting data bases for a variety of applications. A single DBMS is used to manage the data bases for a variety of application environments; the generalized facilities of the DBMS are shared in all application environments. This is in contradistinction to fully tailored, stand-alone application systems which maintain their own internal data bases.

Data base management systems are designed to accomodate collections of information which conform to the restricted meaning of "data base" provided above. In sum, the important features which accentuate the nature of DBMS applications are as follows:

1. There is a large amount of data; these data are interconnected. This distinguishes the DBMS environment from a conventional file system, where data interconnections are

minimized, and structural complexities avoided.

2. The data are primarily formatted, as opposed to textual. This differentiates the DBMS environment from information retrieval systems, such as computerized libraries and text processing facilities.

3. The data base has a substantial degree of inherent structure and uniformity. This means for example that there are typically a number of data objects of each type and that exceptions to the rule do not dominate. In particular, a DBMS does not attempt to solve the complete "knowledge representation" problem, as defined by researchers in the field of artificial intelligence [Bobrow 1977a, Bobrow 1977b, Brachman 1976, Szolovits 1977].

4. There is typically a spectrum of types of users requiring access to a data base. A typical DBMS is not solely oriented to programmers, naive nonprogrammers, nor to some specific applications program. The various users of a data base have different needs, abilities, and levels of computer expertise, to which a DBMS must respond.

1.3. Data Models

The essential role of a DBMS is to act as an intermediary between the application programs or end users who wish access to a data base, and the physical data. In so doing, a DBMS provides two principal facilities:

1. The DBMS is based on a *data model*, which is a formalism in which the logical structure of a data base is defined [Sibley 1977a]. A data model is a collection of data

structure types. These data structure types are instantiated to construct a *schema* for a particular data base; a schema is thus the description of the structure of a particular data base. All information in a data base is logically organized in terms of the structures of the data model.

Typically, data models have a very limited number of data structure types. This is true of the most popular data models: the hierarchical model, in which the records of a data base are organized into trees; the network model, in which the data is organized into records interconnected by a graph structure; the relational model, in which a data base is organized in terms of tables [Date 1977, Kerschberg 1976c].

The specification of a schema is described in the *data description/definition language (DDL)* of a DBMS. A DDL provides the mechanism whereby the data structure templates of a data model are used to define the structure of a specific data base.

2. A vocabulary of general purpose data base retrieval and update operations is provided. The *data manipulation language (DML)* of a DBMS facilitates the invocation of these primitives, and their combination to allow the specification of user-defined data base transactions. A DML specifically allows selected fields of selected records to be retrieved from a data base, changes to be made to existing records (e.g., by assigning a new value to some field), new records to be added to a data base, etc. Typically, the DML may be used as a stand-alone query and update language, or it may be used in conjunction with a host general purpose programming language (such as

COBOL, PL/I, FORTRAN, etc.).

According to the ANSI/SPARC specifications for the architecture of data base management systems [ANSI/X3/SPARC 1975], there are three levels at which a data base is described:

1. At the middle level is the *conceptual schema*, which should capture the meaning of the information in the data base. A conceptual schema is a specification of the data base phrased in terms of the abstract entities and relationships in the application environment. It is at this level that the data model of the DBMS is used.

2. On top of the conceptual schema are the *external schemas*, which represent the views of various users of a data base. These external schemas are a logical reorganization and/or reformatting of the information in a conceptual schema, and are used to tailor the information in a conceptual schema to the needs and tastes of specific users. External schemas are constrained to be compatible with their underlying conceptual schema, e.g., in that operations phrased in terms of an external schema must be mapped into operations in terms of the conceptual schema.

3. The third and lowest level of data base description is the *internal schema*, which describes the physical organization of a data base. At this level, the physical data structures and access methods which are used to store and access the information in a data base are specified.

The utility of separating the conceptual and internal schemas has been strongly emphasized during the past several years of data base research. For the reasons noted above, it is important that a DBMS be oriented to providing a user interface stressing the logical, semantic structure of data, rather than its physical organization. Recent work on "data independence" [Codd 1974c, Date 1971a, Date 1971b, Date 1974] has focused on isolating the conceptual (logical) data model from issues of physical representation; this allows the details of the physical storage representation to be altered without affecting the view of the database presented to users and application programs. In a similar vein, the great importance of separating the behavior of a "data abstraction" from its implementation has been noted for its impact on software quality, reliability, and verifiability [Liskov 1974, Liskov 1977]. In this work on "abstract data types", the emphasis has been on providing operational characterizations of data abstractions [Hammer 1976e], and on suppressing irrelevant detail and superfluous data artifacts introduced by implementation techniques.

1.4. A "Semantic" Data Model

The conventional data models used to structure the data in contemporary data base management systems are in many ways unsatisfactory for modelling data base application environments. The features they provide are too low-level and computer-oriented. A user is required to think in terms of representation rather than in terms of meaning. The *semantics* of a data base that is defined in terms of these mechanisms are not readily apparent from the schema; instead, the semantics must be separately specified by the data base designer and consciously applied by the user.

In this thesis, we present the design of a higher level, conceptual data model, which allows a data base designer to directly incorporate a large portion of the semantics of the application environment in the data base schema. This *semantic data model (SDM)* serves as a natural application modelling mechanism to capture and express the structure of a problem domain. Specifically, we focus on the use of this data model in improving the understandability and accessibility of computerized data bases.

As it is intended that the features of the SDM correspond to the principal intentional structures naturally occurring in contemporary data base applications, the keystone of the methodology employed in designing the SDM was the identification and classification of the most important and frequently occurring constructs in typical data base applications. That is, we sought to discover a set of general structures that repeatedly occurred in different problem domains and that *effectively* expressed the essential meaning of each domain. These constructs served as the bases for the features of the SDM. Moreover, the design of the SDM was very closely tied to a detailed analysis of the most important semantic problems of conventional data models.

As should become clear later in this thesis, the SDM provides a rich but limited vocabulary of data structure types and primitive operations for the manipulation of those structures. The goal of the SDM is to provide for sufficient modelling flexibility to naturally accomodate most data base application environments, while at the same time limiting the number of ways a given fact can be modelled. These limitations are imposed in order to make the task of designing an SDM data base tractable, to make it possible for a user to readily understand the meaning of an SDM schema, and to facilitate implementation. We strive in the SDM to strike a balance between semantic expressibility and complexity.

1.5. Applications of the SDM

As noted above, the SDM is designed to enhance the effectiveness and usability of data base management systems in several ways:

1. An SDM description of a data base serves as a formal specification, documentati ·, and description of a data base. It provides an accurate, concise, and comprehensive statement of a data base's meaning. SDM data bases are to a large extent self-documenting, in the sense that the description and structure of a data base are expressed in terms which are close to those used by users in describing an application environment. This makes the SDM useful as part of the permanent documentation of a data base and as a tool for communicating the meaning of a data base among its users, and between the users and the DBMS itself. Moreover, an SDM schema can serve as a communications medium between an application specialist and a data base designer, precisely describing the system for which a data base is to be constructed. Note that these uses of the SDM are in some sense unrelated to the issue of computerized data bases; we believe that the SDM can be a useful mechanism for precise interpersonal communications about the structure of a problem domain.

2. The SDM supports powerful user interface facilities, which can accomodate users with different needs and abilities. Such user facilities can be constructed as "front-ends" to

existing data base management systems, or used as the basis for the design of the user interface of new data base management systems. The principal reason the SDM can provide these facilities is that SDM data bases capture *information in a form* that stresses the meaning of data rather than its representation in the computer.

As we shall see, facilities for expressing derived (conceptually redundant) information are an essential part of the SDM; derived information is as prominent in an SDM schema as is primitive data. Derived information in an SDM schema has an important impact on the user: the presence of derived information in the SDM supports multiple ways of viewing the same data (external schemas). Moreover, since the uses of a data base often change as applications grow, a data base schema must itself adapt to its uses. *In terms of the user, this means* that it is important to integrate the most common types of retrievals (data selections) into a schema as derived information. These pieces of derived information can then serve as building blocks for related and potentially also common interactions: it is likely that if a derived data item is used often, then it may be helpful in the description of other transactions. Thus, a data base can be made to adapt to optimize the users' views according to current needs.

In particular, the SDM provides a basic for a high level, semantics-based data base query and update language. Such a language, called the SDM "interaction formalism" has been designed, and is described later in this thesis. As we shall see, this language is intimately related to the SDM, in that it provides rich but limited *set of* built-in data base operations, and allows user-defined transactions to be specified in terms of these primitives. The combination rules are simple, but the vocabulary of primitives allows a good deal of flexibility in describing data base retrievals and modifications.

A prototype interactive user interface facility based on the SDM has also been designed and implemented. This "interaction formulation advisor" (IFA) guides a user through a data base and the process of formulating a query or update request. It assumes that the user is largely naive of the data base content and structure, and that the user has very limited experience with computerized data bases. The IFA is not merely a passive data base interrogation tool; rather, it is based on a specific structured, stepwise methodology for expressing data base queries. It is by means of this methodology that the IFA can assist naive users. The IFA is also closely guided by an SDM schema for the data base with which the user is working. A prototype IFA is currently running, and is described later in this document.

3. The SDM provides a basis for tools to aid the data base and application designer as well as the naive user. In particular, the SDM can be used as a tool in the data base design process. This type of SDM data base design tool would supply a data base designer with a collection of templates, which aid in the identification of relevant information in a data base application environment. This can serve to aid the designer in organizing relevant information, relating it to its possible applications, and translating it into lower-level representational structures to be implemented. An important advantage of designing a data base in this way is that a rather precise and

complete documentation of its contents and meaning is readily available. This is especially useful for application programmers, who need to determine what information is in a data base, what it means, and how it is logically organized. Furthermore, since derived information is directly accomodated in an SDM data base, the data base's structural description can track the evolution of its uses.

It is important to note that the use of the SDM is not dependent on the successful implementation of a new data base management system that directly supports it. There are already a large number of basically satisfactory data base management systems in use today, which represent a considerable investment by their developers and users; the SDM can be effectively used in conjunction with such systems to enhance their effectiveness and usability.

1.6. Thesis Overview

The remainder of this thesis is organized into two main parts. The first part, consisting of chapters 2 through 4, focuses on the SDM and its use as a modelling tool. Chapter 2 describes in detail the principal semantic modelling problems associated with conventional data models, and reviews related work. Chapter 3 then provides a detailed description of the SDM. In chapter 4, a discussion of the modelling capabilities of the SDM is provided, and the SDM is reviewed as a solution to many of the problems of conventional data models described in chapter 2.

The second major portion of this thesis examines user interfaces to data base management systems, and focuses on user interface tools based on the SDM (chapters 5 through 7). In chapter 5, a powerful, semantics-based data base query and update language for the SDM (the "interaction formalism") is described. Chapter 6 contains a review of user interface considerations for data base management systems; in this chapter, recent work on improving user access to data bases is discussed and the principal unsolved user interface problems are analyzed. Finally, in chapter 7, the details of the interaction formulation advisor are presented. In a sense, chapter 7 is an "acid test" of the SDM, in that it shows that the SDM provides a unique mechanism which can support a new type of data base user interface tool.

2. SEMANTIC PROBLEMS WITH CONVENTIONAL DATA MODELS

Before embarking on our discussion of semantic problems with conventional data models, we must first outline the most common and popular data models used in contemporary data base management systems.

2.1. The Network and Hierarchical Data Models

The *network data model* [Taylor 1976] is used by many commercial data base management systems (e.g., IDS [Honeywell 1972], IDMS [Cullinane 1975a, Cullinane 1975b], and TOTAL [Datapro 1972b]), and has received a good deal of attention over the past several years. The archetypal network data model is that proposed by the Codasyl DBTG committee [Codasyl 1971] and advocated by Bachman [Bachman 1973].

In the Codasyl DBTG network model, all information in a data base is structured in terms of "record types" and "set types". Record types are collections of records, each of which is a collection of fields, whose values are character strings or numbers (or groups of these). "Set" types interconnect record types. ("Set" here does not mean mathematical set.) Each "set" type has an owner record type and a member record type; instances of the record type connect an owner to its members. A given record type can have more than one owner, so that complex interrecord network connections can be formed. Other semantic constraints can be captured in a Codasyl schema via several additional mechanisms. For example, one can declare the member record type R in set type S as "mandatory", which means that once a record R1 in R is inserted into an occurrence S1 of S, R1 may not be removed from S1 without actually deleting R1. Similarly, if R is declared as "automatic", each newly created record R1 will automatically be inserted into some set S2; the identity of S2 is determined by a "set selection clause" provided by the designer.

The goal of the network data model is to structure data in the form of a graph. Entities in the application environment and their attributes are captured via record types, and set types are used to connect related records. Via set types, a network schema contains an explicit specification of which interrecord paths make sense. Set types accomodate one-to-many associations, and many-to-many associations can be handled by introducing a dummy relationship record.

For historical and other reasons, the *hierarchical data model* [Tsichritzis 1976b] is a very popular one; commercial implementations of hierarchical data base management systems include IMS [IBM 1975a, IBM 1975b] and System 2000 [Datapro 1972a, MRI 1972]. In the hierarchical data model, records are organized in the form of trees; hierarchies are a special case of networks where each record type in a data base has exactly one superior/owner record type.

2.2. The Relational Data Model

In addition to the network and hierarchical data models, the *relational data model* is increasingly popular for use in data base management systems. The relational data model was introduced by Codd [Codd 1970]. The relational model was initially most popular among data base researchers, but is now enjoying increased acceptance in industry.

Examples of relational data base management system implementations include: INGRES [Held 1975a, McDonald 1974a, McDonald 1974b], MACAIMS [Goldstein 1970], PRTV [Hitchcock 1976, Todd 1976], RDMS [Steuert 1974], RISS [McLeod 1975], System R [Astrahan 1976], and ZETA [Brodie 1975, Czarnik 1975].

A *relational data base* is defined to be a collection of normalized *relations* (relations in first normal form [Codd 1970]), and a collection of *domains*. (The relations present in the data base are specifically called *base relations.*) A normalized relation may be viewed as a table, wherein each *row* of the table corresponds to a *tuple* of the relation, and the entries in a *column* (*attribute*) belong to the set of values constituting the underlying domain of that column. An *entry* is the value in some particular column for a given row of a relation. The domain underlying a column consists of precisely those values which can appear as entries in that column; every value in the underlying domain is a plausible entry in that column. The special value *null* is used to indicate that no data item is present in some column of a tuple; a null data item means that the information is not relevant or is not known.

In most existing relational implementations, only machine-oriented domains are provided: integer, floating point number, varying length character string, etc. Alternatively, a separate domain definition facility can be provided, which allows the specification of precisely which atomic data values constitute a domain [McLeod 1977a]. In this way, it is possible to define domains specific to a given application environment, such as names of ships, dates, geocoordinates, and so forth.

In conjunction with the data structure "relation", a relational DBMS must provide a collection of operations for manipulating the relations in a data base. The operations provided must support data base schema creation and modification, data base query/retrieval, and data base update. Figure 2-1 contains an example set of data base operations that might be used in a relational DBMS; the set presented therein is similar to the relational algebra [Codd 1970]. The operations shown indicate facilities for both data definition and manipulation; when combined with rules for combination, a combined data definition and manipulation language may be obtained. Examples of existing relational data base data definition and manipulation languages include SEQUEL [Chamberlin 1974, Chamberlin 1976c], QUEL [Held 1975a], and Query by Example [Zloof 1975a].

The great deal of recent work and interest in the relational data model and associated developments serves as a metric of the increasing recognition of the importance of logical views of data and associated high level data base interaction facilities. We shall discuss the relational model first and in the most detail, as it is widely accepted as a major research advance in the area of data modelling and user interfaces to data base management systems. The relational model will thus serve as a basis for analyzing modelling and user problems with conventional data models. Therefore, we first describe modelling with the relational data model and its associated problems; later, we return to the network and hierarchical data models and consider their advantages and problems vis-a-vis data modelling and user interfaces.

2.3. An Example Application Environment

In order to provide a particular context for our detailed discussion of data modelling and user interaction with a data base, we shall adopt a realistic example of an application environment. This nontrivial application environment is the "tanker monitoring application environment" (TMAE). It involves the control of the flow of traffic of ships, particularly those with potentially hazardous cargoes, such as oil tankers, into and out of U.S. coastal waters (such as might be used in the prototype "Marine Safety Information System" being developed by the U.S. Coast Guard). It should be noted that our purpose here is not provide a comprehensive solution to the problem of handling hazardous ships; rather, we intend to use the TMAE as a rather extensive, realistic, and important data base application environment. The TMAE will also be used later in the discussion of the SDM and the IFA. (For variety, a few aspects of the TMAE discussed here are different from those discussed in the context of the SDM).

2.4. A Relational Data Base for the TMAE

The relational data base we define to model the TMAE is called the "tanker monitoring relational data base" (TMRDB). Figure 2-2 contains a description of the TMRDB. Figure 2-2a contains a relational schema for this data base, viz., a description of the relations. This includes the name of each relation, as well as the name of each column of the relation. For simplicity, we have omitted the specification of the underlying domain of each column. (Figure 2-2b contains a list of the domains for the TMRDB, for reference.)

The TMRDB illustrates a number of problems with the relational data model, as discussed below. However, it was not constructed to be a deliberately poorly designed schema. Rather, the problems it evidences are problems inherent with the relational data model, and these problems are not easily patched within the confines of the features of the relational model. For example, to illustrate the fact that both data bases in third normal form [Codd 1971b, Codd 1971c] and those not in third normal form suffer from problems, part of the TMRDB is in third normal form, and part of it is not. (Third normal form relations are those which do not exhibit certain types of update and deletion anomalies, caused by functional dependencies that do not correspond to the basic structure of the relation.)

In order to aid our subsequent analysis, we shall briefly describe the TMRDB. Significantly, our explanation of the TMRDB is intended to be typical of the way a relational data base designer might explain it. Thus, in addition to serving as a description of the TMRDB, this illustrates the problems in using the relational data model as a formal data base specification mechanism.

The following describes the TMRDB relations:

1. Relation SHIP contains a tuple for each relevant ship; it is used to keep track of the name, type (e.g., "merchant", "naval", etc.), hull number, radio call sign, country, engagement (current activity), maximum speed, and fuel type of each such ship, as well as whether or not a medical doctor is currently on board the ship.

2. ASSIGNMENT has an entry for each assignment of a captain to a ship, and records

all such assignments in the history of the data base. The name of the captain and ship are recorded, as are the dates the assignment starts and ends.

3. CARGO records the quantity of cargo currently aboard a given ship; the ship is identified by its name.

4. Relation PORT records the name, location (geocoordinate), and country of each relevant port.

5. PORT_OF_SHIP contains a tuple for each ship recorded in relation SHIP, and establishes the identity of the home port of that ship. The column Name contains the name of the port, and Ship_name contains the name of the ship.

6. BANNED_SHIP contains the name and effective date of banning of each ship (in SHIP) that has been banned from entering U.S. coastal waters. (Only the most recent ban of a ship is recorded.)

7. Relation POSITION_REPORT contains information on each report of position/status received from a ship. Position reports can be filed by ships or by convoys; in the latter case, the convoy implicitly reports the position of each of its member ships. If the position report is for a convoy, Ship_name is null, and if it is for a ship, Convoy_name is null. The relation keeps track of the date and time of the report, whether or not the ship or convoy is in port, and the name of the port (if the ship or convoy is in port) or the geocoordinate of the present location of the ship or convoy (if it is at sea).

8. REPORT_PRECISION contains further information regarding a position report. It

logically related to POSITION_REPORT on columns Ship_name, Convoy_name, Date, and Time, which together constitute a unique identifier for both POSITION_REPORT and REPORT_PRECISION. Recorded here is the type of precision (circular or elliptical), and the circular radius or the major axis, minor axis, and inclination of the ellipse.

9. Relation SAILING_PLAN records information on the plan filed by a captain before his ship or convoy leaves port. It contains the name of the ship or convoy (again, the other column has a null value), and the date filed. (A single column might be used for both cases, which can contain either the name of a ship or the name of a convoy.)

10. SAILING_PLAN_STOPS carries further information on sailing plans: the name and scheduled arrival date for each port to be visited. The relation is logically linked to SAILING_PLAN on Ship_name or Convoy_name, and Date_filed.

11. The relation COMMERCIAL_SHIP records the name and weight of each ship that is commercial.

12. MILITARY_SHIP records the name and displacement of each military ship. (By convention, commercial ships have a weight and military ships have a displacement.)

13. OIL_TANKER records additional information on each ship that is an oil tanker: its date of last inspection and its hull type (e.g., "double hull"). As for relations COMMERCIAL_SHIP and MILITARY_SHIP, the ship name is used to uniquely identify each tuple of OIL_TANKER; all ships are assumed to be recorded in relation SHIP.

14. Relation INCIDENT records all ship accidents. An incident number is established, and the date, time, and (narrative) description of the incident are recorded.

15. INCIDENT_SHIPS is linked to INCIDENT on incident number, and contains the name of each ship involved in the incident.

16. INSPECTION records the inspection of the named ship; the name of the inspector, the date, the violations, and the disposition of the inspection are recorded. The violations are recorded in textual form.

17. Relation INSPECTOR lists the names of all inspectors.

18. CONVOY is used to keep track of which ships are in which convoy. Convoys are given a name, and the ships therein are listed by name. For example, for a convoy with three ships, there will be three tuples in relation CONVOY.

19. The relation SHIP_TYPE records information about the type of cargo that can be carried on each type of ship. The type of ship is related to the type of ship indicated in relation SHIP.

In addition to the base relations, i.e., those stored in the data base, there can be (multiple) *views*. A view is a virtual relation that is derived from base relations and other views [Astrahan 1976, Chamberlin 1975, Dale 1977b, Eswaran 1976b, Klug 1977, Stonebraker 1975b, Summers 1975b, Zaniolo 1977]. Many proposed or implemented relational data base management systems have some kind of view handling capability. Views correspond to the "external schemas" defined in [ANSI/X3/SPARC 1975]. Views can be defined by retrieval expressions that materialize the relation constituting the view; the definition of a view is thus very similar to the formulation of a query. Examples of views of the TMRDB are given in Figure 2-3. We briefly explain these views here:

1. The view (derived relation) CURRENT_SHIP_CAPTAIN is provided in order to associate the name of each ship with the name of the officer who is currently assigned as its captain. The definition of this view can state that to define the view, one finds, for each tuple in SHIP, the tuple in relation ASSIGNMENT with the most recent Date_start that has a Date_end today or later for that ship, and then extracts the Captain and Ship_name column values from that tuple.

2. SHIP_WITH_FLAG is exactly the same as relation SHIP, except that the column Country in SHIP is renamed as Flag in SHIP_WITH_FLAG.

3. View CURRENT_SHIP_POSITION establishes the current location of each ship (as recorded in relation SHIP). If it is in port, then the port is recorded and Geo_coordinate is null, otherwise Port_name is null and Geo_coordinate contains the location of the ship. The definition of this view is rather complex, in that we must determine for each tuple in SHIP, whether or not that ship is in a convoy (using relation CONVOY). If it is not, we find the most recent tuple in POSITION_REPORT for that ship and place the Ship_name, Geo_coordinate, and Port_name values into the view. If the ship is in a convoy, we do the same but instead find the most recent tuple in POSITION_REPORT for the convoy in which the ship is currently engaged.

4. The view OIL_TANKER_MISHAP contains all information on oil tanker mishaps,

i.e., on incidents involving ships that are oil tankers. To define this view, we find each tuple in INCIDENT that represents an incident involving an oil tanker; this is accomplished by finding the corresponding tuple in INCIDENT_SHIPS by matching on incident number, taking the ship name and finding the appropriate tuple in SHIP, and seeing if that ship is an oil tanker (by examining the value of attribute Ship_type of SHIP). The contents of each qualifying tuple in INCIDENT are placed into OIL_TANKER_MISHAP, along with the name of the ship (oil tanker) involved in the incident.

5. SHIPS_DUE_FOR_INSPECTION contains information on all ships that are currently due for inspection. This view is defined by placing in it each tuple of SHIP whose most recent corresponding tuple in INSPECTION has a date that differs from the current date by more than the allowed amount; the appropriate tuple in INSPECTION is found by matching on common value of Ship_name (for the tuple in SHIP) and then extracting the tuple with the most recent value in column Date for that ship.

6. The view CONVOY_SIZE contains the name and number of ships is each convoy. It is defined by grouping the tuples in CONVOY by common value of Convoy_name and counting the number of values of Ship_name in each group.

7. SHIP_AT_SEA contains the name of each ship currently at sea. This view is defined by finding the corresponding tuple in view MOST_RECENT_POSITION for each ship therein; if the ship is indicated to be at sea, then the name of the ship is placed

into the view.

8. View DANGEROUS_SHIP contains the name and type of each ship currently considered dangerous; a ship is termed dangerous if it currently banned from U.S. coastal waters. DANGEROUS_SHIP includes the name of each ship that appears in BANNED_SHIP; the type of the ship is also included in the view, as determined by the appropriate tuple in relation SHIP.

2.5. Semantic Information Representation in the Relational Data Model

Having introduced the relational data model and the TMRDB, we are now in a position to discuss the representation of semantic information in a relational data base. We emphasize how semantic information is presented to the user via the relational data model. There are a number of ways in which semantic information is captured in a relational data base. Of course, the specific way selected to model a particular fact has a significant impact on the user, both in terms of his understanding of the data base and in how he accesses it. Specifically, the following methods of capturing information are most significant:

1. Values in the same tuple of a relation are associated. This *horizontal connection* logically links the atomic data values appearing in the columns of the tuples. For example, the TMRDB relation SHIP has columns Ship_name and Hull_number, thereby associating the name of a ship to its hull number.

2. Tuples in a relation that share a common value in one or more columns are logically grouped together. This *vertical connection* logically links together tuples that share some

common property. For example, each group of tuples in INSPECTION that share a common value of Ship_name represents the set of inspections for the ship with that name.

3. If two columns from two relations share a common underlying domain, one or more tuples in a relation are logically linked to one or more tuples in another relation; this logical link is a *join connection*. The link can be one-to-one, one-to-many, or many-to-many. For example, a ship is matched with its home port via the join of SHIP and PORT_OF_SHIP on common value of column Ship_name. (A join can also logically link tuples in the same relation, e.g., matching a ship to its sister ship via a column named Sister_ship_name.)

4. Semantic information is carried by the placement of a particular piece of information in a specific relation. (*residence relation selection*). There is often a choice concerning in which relation a new tuple is to be placed. For example, the creation of a new oil tanker may be modelled by adding a tuple to relation SHIPS, or it may be captured by adding a tuple to relation OIL_TANKERS, or both.

5. *Relation naming* provides a vehicle for capturing semantic information in a relational data base. Relation names may carry information about their contents, e.g., the relation named COMMERCIAL_SHIP contains precisely those SHIPS that are commercial.

6. *Column naming* provides yet another way to express semantic information. Columns can be named to describe their meaning, e.g., naming columns "Ship_name" and "Port_name" instead of just "Name". Information is also carried by the identity of the

abstract set of values from which the current value in some column of a tuple is selected (the underlying domain of a column). For example, the user may conclude that if two columns have the same underlying domain, their values can be compared [McLeod 1977a], e.g., the column Port_name of relation PORT and the column Name of relation PORT_OF_SHIP can be compared because they both contain names of ports. The issue of which columns of relations are comparable is an important one, in that a join (cross link) of relations normally should be performed on comparable columns if it is to make sense, e.g., joining SHIP and PORT on Port_name and Ship_name is almost meaningless.

In addition to the above, there are several other ways in which semantic information can be captured in a relational data base, if the basic relational data model is somehow supplemented:

1. Additional "semantic integrity constraints" can be used to model additional semantic information in an application environment. For example, subset constraints between columns of relations, uniqueness/keyness constraints, and the like introduce capabilities for modelling additional types of semantic information [Eswaran 1975, Hammer 1975a, Hammer 1976d, Stonebraker 1974b].

2. Additional semantic information about a data base may be captured within applications programs and "canned" data base transactions (transactions whose descriptions are saved and repeatedly executed). Such transactions may for example contain error checking routines and may perform manipulations of a data base which

are based on additional "knowledge" of its semantic structure. For example, a transaction may automatically insert a corresponding tuple into SHIP when a new tuple is inserted into OIL_TANKER.

3. Views can similarly capture semantic information; a view materialization expression which defines a view is similar to a transaction and it therefore may capture additional semantic information about a data base. For example, the definition of the TMAE view CURRENT_SHIP_POSITION captures information about how the current position of a ship is determined, i.e., it captures knowledge of the meaning of "current ship position".

The most obvious and significant problem with capturing semantic information in these ways is that the user has to deal with various mechanisms to free that information. The user must be conscious of the fact that the information he wants may be obtainable via a horizontal or vertical connection, via a join, by examining the names of relations and columns, or by some other method. The level of the mechanisms provided is too low to be understood and manipulated by many users.

2.6. Semantic Problems of a User View Based on the Relational Data Model

With the above indication of how semantic information is captured in a relational data base, and with the explanation of the TMRDB, we can now proceed to identify specific modelling problems associated with the relational data model. That is, we can examine the shortcomings of the relational model in:

1. making the meaning of a data base apparent to its users,

2. allowing a data base to be explained to a user in a concise, structured manner,

3. enabling a user to easily select the information he requires from a data base.

We note that there is a rather wide spectrum of problem types, that range from general to specific in nature, as follows:

1. The relational data model, like other conventional data models, is based on *syntactic data structures*. The data base designer and user must effect a conscious transformation from his view of application-oriented entities to their representations. In other words, the designer and the user are forced to think in terms of data, rather than reality; they must deal with the representations of things (or with information about them), rather than with the things themselves. The fact that the single, supposedly general-purpose data structure "relation" is used to model all relevant information in an application environment makes the user's view simple. Unfortunately, it also causes semantic overloading, i.e., the same structure, the relation, is used for many purposes, such as:

   a. to record the existence of entities, e.g., ships (by the existence of appropriate tuples in relation SHIP),

   b. to describe properties of an entity, e.g., the name and length of a ship (by appropriate attributes in relation SHIP),

   c. to model associations between entities, e.g., assignments of captains to ships (via relation ASSIGNMENT),

   d. to describe collections of entities, e.g., convoys of ships (using relation

CONVOY).

This semantic overloading makes it difficult for a user to extract the meaning from a relational schema. For example, he may find it quite hard to tell which of the above listed uses is being made of a given relation.

2. In the relational data model, *denotation by selected name* plays a central role in a data base. It is necessary to use a selected name to refer to entities in a data base. The main implications of this on the user interface are:

a. It is necessary to cross reference explicitly from one relation to another. For example, a user must explicitly link a tuple in SHIP to a tuple in ASSIGNMENT to find the captain of the ship. This indirectness causes problems that are somewhat analogous to those seen in cell-oriented programming languages, where data objects are referenced through named storage locations in lieu of operating directly on the objects themselves. The requirement for a user to do explicit cross referencing among relations results in a heavy reliance on join connections (links across relations on common values) in the relational data model; when combined with the problems of semantic overloading mentioned above, this makes it hard for a user to connect related information.

b. The distinction between changing an entity and changing a name used to denote it is muddled, since names are actually used to represent entities. For example, when the name of a ship is changed, an appropriate update may be made to a tuple in relation SHIP; however, the columns of other relations which are logically connected to SHIP, such as POSITION_REPORT, now must also be changed. This problem would be avoided if column Ship_name of POSITION_REPORT has ships as values, rather than ship names; unfortunately, this is not possible in the relational data model. These difficulties stem from a failure to recognize the importance of the distinction between an entity and its name. It is well known to programming language designers that failing to adequately accomodate this distinction can be a source of much confusion.

c. For reasons of economy, it becomes necessary to organize the data base so that all references to an entity are made through a standard attribute (often called a "key"). For example, a major portion of the TMRDB is oriented to ship names. This makes it difficult to refer to entities via other names. For example, it is hard to refer to ships via hull numbers: if a user is interested in BANNED_SHIPS, it is simple for him to obtain the name of that ship, as it is directly recorded in the tuples of BANNED_SHIPS; however, if he is interested in the hull numbers of banned ships, he must reference relation SHIP to establish the correspondence between ship names and hull numbers.

3. The relational data model is based on *atomic value sets*; that is, all columns have values that are atomic (nonhierarchic). This problem is closely related to the problem of denotation by selected name. The nonhierarchic, nonrecursive nature of relations simplifies the data model, but causes several significant problems:

a. It makes it difficult for a user to travel easily to a related entity from one he has in hand. A user must explicitly cross link from one relation to another by describing an explicit join.

b. It makes it impossible to use precise semantic collections as value sets of attributes (columns of relations). For example, we would like the underlying domain of a column Ship of INSPECTION to be OIL_TANKER (or, at least, the names of oil tankers); but we must use the domain SHIP_NAME as the value set here. The problem is that it is not possible to specify underlying domains that are related to information in the data base, e.g., it is not possible to state that the underlying domain of Ship_name of INSPECTION is the set of all current values in column Ship_name of SHIP.

c. It requires a nontrivial additional facility to allow the definition of abstract sets of atomic data values [McLeod 1977a]. Even if such a facility to allow the definition of the sets of values underlying columns is provided, it is still not clear when domain definition capabilities should be used and when relations should be used. For example, to model the set of names of inspectors (a subset of the set of person names), should an appropriate domain be defined or should relation INSPECTOR be used (which, for example has one column with underlying domain PERSON_NAME)?

4. Only *single-valued attributes* are permitted in the relational data model, so that attributes whose values can be collections of entities are difficult to handle. These "repeating groups" are a natural structure to users, but must be expressed in the

relational model in one of the following ways:

a. An extra relation can be introduced. This relation accomodates the multi-valued attribute A of relation R in the following way: it has a column by which it is logically linked to R; it has another column which contains the values of the (potentially repeating) values of A. The link between these two relations can be either explicit or implicit, as the following examples indicate:

i. For sailing plans, we have relation SAILING_PLAN, plus another relation for the repeating attribute of scheduled stops; the repeating attribute is handled by introducing relation SAILING_PLAN_STOPS, which is implicitly linked to SAILING_PLAN by common value of Ship_name and Date_filed.

ii. For incidents, relation INCIDENT is linked explicitly to INCIDENTS_SHIPS by means of introducing the attribute Incident_number.

b. A multi-valued attribute can be accomodated by simply concatenating its values. However, this approach is quite unnatural, and it requires string manipulation facilities to be used. Because of the consequent unnecessary manipulation complexities, this approach is of limited practical value.

5. It has often been stressed by data base researchers that relational data bases should be *nonredundant*; the has resulted in a good deal of work on relational "normalization" [Beeri 1977, Bernstein 1975a, Bernstein 1975b, Bernstein 1975c, Bernstein 1976, Codd 1970, Codd 1971b, Codd 1971c, Fagin 1976, Fagin 1977a, Fagin 1977b, Kent 1973, Sharman 1976]. The principal goal of this work is to record each fact in only a single place. Much of

the motivation for normalization comes from the desire to avoid certain types of anomalies which result when an "unnormalized" relational data base is updated [Codd 1971b, Codd 1971c]; these anomalies involve, among other things, the necessity of side-effects when tuples are updated or deleted.

Normalization is hinged on the concept of a functional dependency between columns of a relation; a functional dependency exists between columns C1 and C2 of relation R if there is at most one value of C1 for each value of C2. Relations in "third normal form" (and higher order forms) conform to certain kinds of restrictions on the functional dependencies between their columns. Relations in these normal forms are free from certain kinds of undesirable update anomalies.

Normalization is thus intended to accomplish the following:

1. It eliminates certain types of update anomalies.

2. It increases the ability of a relational data base to capture certain types of semantic information. (Needless to say of course, functional dependencies can capture only a portion of the desirable semantic information [Bernstein 1975a, Bernstein 1975b, Bernstein 1975c, Bernstein 1976].)

3. It allows a relational data base to be organized in such a way that logical redundancy is reduced. However, normal forms eliminate only limited kinds of redundancies. In practice, there are a tremendous number of ways in which logical redundancy can appear in an application environment.

In consequence of the third point above, we believe that it is not feasible to eliminate redundancy from data bases. On the contrary, it is not even desirable for a data base user to see a nonredundant data base. Application environments are inherently redundant, and to accurately model the semantics of an application environment by a data base, a redundant schema is a great help [Bubenko 1977a, Meltzer 1976]. The relational and other conventional data models have great difficulty in integrating redundant (derived) information into a data base schema.

6. The relational data model does accomodate redundancy and derived information to some extent by means of *superimposed views*, but this information is not integrated into a data base schema. In effect, views can serve as retrieval transactions on a data base, since they derive information from it. For example, view CURRENT_SHIP_CAPTAIN may be established if users commmonly need to associate a ship with its current captain. However, there are several serious problems with superimposed views:

a. Views place semantic information outside a data base schema. This means that it is much harder for the data base designer to balance the potentially conflicting needs of the views. The onus is also on the user to locate a view appropriate to his current needs. This may be quite difficult for him, just as it may be hard to find a relevant stored transaction, or to find a relevant program in a program library.

b. Since a view is defined by a materialization expression which is essentially a data selection transaction (query), views can accomodate only information which is derived from data in the data base. Views cannot include additional "primitive" information, which is to be explicitly entered by users. For example, in the TMAE it might be necessary to record additional information on ships at sea (as opposed to other ships), but view SHIP_AT_SEA is derived from SHIP and can have only derived information in it.

c. The proposed view mechanisms for relational data bases [Astrahan 1976, Chamberlin 1975, Dale 1977b, Eswaran 1976b, Klug 1977, Stonebraker 1975b, Summers 1975b, Zaniolo 1977] all include limited capabilities for defining derived information. They allow derived relations to be defined as row and column subsets of the tuples of relations the data base, and as joins of data base relations. Thus these facilities do not focus on the different semantic ways in which derived information can be defined. For example, we should like to allow a user, given a ship, to refer to all inspections of it. In this case, a derived attribute of *ship entities* is needed; the value of this attribute is calculated by examining the attribute of inspections that specifies the identity of the ship being inspected.

d. The facilities provided for designing views do not strongly encourage good design. It is all too easy to define a view inappropriate to the current and future uses of a data base. For example, view SHIP_AT_SEA has been defined to provide the name of each ship banned from U.S. coastal waters, but it is difficult to use this view to find the hull number of each such ship.

The important point is that derived information should be integrated into a data base rather than superimposed on top of it.

7. The relational data model suffers from the problem of *disjoint structures*. Relations

and the information in them are isolated, and are not well connected. Tuples of relations can be linked only by join connections; this severely limits the possible ways of expressing interrelationships between entities. An important consequence of this semantic inflexibility is that tuples are members of only one relation (the "legacy of record technology" [Kent 1976]). This means, for example, that it is difficult to capture the fact that a tuple in the relation for all ships and a tuple in the relation for all oil tankers represent the same entity (ship).

8. The problem of *nonintegrated subtypes* is a consequence of the use of disjoint structures. It is not possible to model the fact that oil tanker is a subtype of ship. (This is the "is a" relationship often referred to in artificial intelligence research on knowledge representation.) Such subtype information is important to the meaning of data. One important aspect of subtypes is the inheritance of attributes; for example, all attributes of ships are inherited by oil tankers, since all oil tankers are ships. Although limited capabilities of subtype definition can be accomplished via the introduction of semantic integrity constraints [Hammer 1975a, Hammer 1976d], this is a somewhat ad hoc solution to an important problem of semantics and accurate modelling of the application environment.

9. There is a set of *rigid attributes* for each relation, which means that the existence of an attribute cannot be conditional on properties of the entity it describes. For example, in relation SHIP_POSITION, we must have both attributes Ship_name and Convoy_name, because some position reports are for ships and some are for convoys.

"Null"s must be used for irrelevant attributes. Alternatively, separate relations could be introduced to accomodate position reports for ships and position reports for convoys; however, if this approach were adopted, it would be difficult to meaningfully relate the two types of position reports (mainly due to the problems of disjoint structures and nonintegrated subtypes).

10. The collection of entities in an application environment into aggregates (groups) is handled in the relational data model by associating the values of a name for the aggregate (*aggregation by name association*). No direct semantic construct is provided to model collections of entities. Instead, aggregates are modelled by using vertical connections within a relation; each aggregate is defined as consisting of those tuples in a relation sharing a common value in a column. For example, convoys are modelled by means of relation CONVOY; each convoy is given a name and the members of a convoy are those ships that share a common value of Convoy_name. Notably, this forces a view of aggregates that is too low in level. It may also require the introduction of an artificial attribute; for example, Convoy_name of CONVOYS may not be a meaningful attribute, but rather introduced only in order to allow convoys to be defined in the data base.

11. In the relational data model, dynamic subsets of collections of entities are described by collecting together their names (*dynamic subsets by name*). A subset of a collection of entities is often modelled by a relation that contains the names of the entities in the subset. For example, relation BANNED_SHIP records the set of ships that are currently

banned from U.S. coastal waters. An important semantic construct, the subset, is handled artificially. Again, the structures a user sees are at too low a level. In the example, the user must also cross reference to relation SHIP to find the attributes of ships present in relation BANNED_SHIPS.

12. The relational data model lacks a capability for relating entities to their types (*artificial abstraction*). For example, it is not possible to capture the fact that relation SHIP_TYPE contains tuples that model types of ships, i.e., that tuples in SHIP_TYPE are type abstractions, whose instances are modelled by tuples in relation SHIP.

13. The relational data model suffers from a *static orientation*: events, activities, and actions [Bubenko 1977a, Bubenko 1977b, Langefors 1974, Langefors 1977] in an application environment are not directly accomodated. For example, the event of a ship sailing into port must be indirectly and artificially modelled by recording in a position report that a given ship is in a particular port. And, when this fact is modelled in this form, it may be hard for a user to find it.

14. The lack of an adequate capability for modelling other types of events, such as those that have a duration (as opposed to instantaneous ones) causes the need for *artificial association entities* in the relational data model. It is often necessary to construct relations whose sole purpose is to relate entities. This is reasonable if the association among entities makes sense as an entity itself. For example, if it makes sense to talk about assignments of captains to ships as entities in the TMAE, then it is not unreasonable to use relation ASSIGNMENT for this purpose. In other cases, it is

sometimes more semantically appropriate to model entity relationships via attributes of those entities. In such cases, an artificial association entity must be defined in the relational data model. If the concept of "assignment" were not relevant in the TMAE, then relation ASSIGNMENT would introduce an artificial association entity. Even when an association entity makes sense, it is better to view it at a higher level, e.g., as a duration event.

15. The representation of (semantic) *information in relation and column names* causes some problems for users, since names are used for so many purposes. This reliance on names is necessary because the relational data model does not provide sufficient means for expressing important semantic information. It is sometimes difficult to name some of the relations that must be introduced, because they do not directly model things in the application environment. Examples include view SHIP_WITH_FLAG, and column Name in relation PORT_OF_SHIP. Moreover, a user cannot express second order interactions (involving relation and column names), e.g., the user cannot find all relations that contain ships (e.g., OIL_TANKER, COMMERCIAL_SHIP).

16. The relational data model incorporates an *inflexible degree of binding* in data modelling. In order to capture the fact that some aspects of entities do not change, it is necessary to distinguish "fixed" and "changeable" data. For example, it is essential to somehow capture the fact that the length of a ship should not change (except to correct a data recording error), while the position of a ship is normally highly changeable. The relational data model does not accomodate this differentiation.

17. Relational data bases tend not to be organized into relations which are natural units for update (*lack of semantic update units*). For example, because multi-valued attributes are not allowed in a relation, an update which changes that multi-valued attribute must affect more than one relation. Many of the problems mentioned above contribute to this lack of semantic update units. As a further example, we note that although the filing of a sailing plan may be a natural unit of update activity, the information to be recorded in this update is scattered into relations SAILING_PLAN and SAILING_PLAN_STOPS. This makes it difficult for update transactions to be defined and understood, and for the DBMS to automatically check that an update is meaningful.

18. *The relational data model is lacking in its ability to provide (multiple) orderings* among the data (*missing ordering capabilities*). Ordering of information (e.g., tuples in a relation) is an important structuring capability, but it is not directly accomodated in the relational data model. For example, it is important to be able to refer to "the last three inspections of a ship"; to accomplish this kind of ordering reference in the relational data model, the user must awkwardly nest predicates on attributes of INSPECTION, or he must manually define an ordering attribute by scaning through INSPECTION for the last three tuples for each ship.

19. Much information about the structure of an application system simply cannot be expressed in a relational schema (*unexpressed semantic information*). The relational data model provides only a limited set of mechanisms with which to express information

about the organization of the problem domain, and some of these are cumbersome to use. Knowledge that cannot be naturally represented in these terms must either be carried explicitly as data in the data base, or be captured by means of an external mechanism such as semantic integrity constraints, or it will simply remain unexpressed. We believe that the more information about the meaning and structure of a data base that is explicitly expressed in its schema, the less the user needs to remember and keep in his mind; user interaction with the data base then becomes simpler and more reliable. The situation is analogous to the move from representation to abstraction in data types as a means of achieving software reliability.

Having now detailed the semantic modelling problems of the relational data model, we consider the network and hierarchical data models. In the discussion, we compare the modelling effectiveness of the network and hierarchical models with that of the relational model, and examine the modelling differences among them.

2.7. Comparison with the Network and Hierarchical Data Models

Before directly addressing issues of data modelling effectiveness and semantic expressibility for the network and hierarchical data models, we note that a good deal of discussion has accompanied the controversy concerning which logical data model is best [Codd 1974c, Codd 1975b, Date 1974]. There are several results of this debate:

1. It is desirable for the user's view of a data base to be at a high level. This means in particular that issues of physical storage structures, access methods, and other

implementation issues should not be included in the logical data base schema. In its classical form [Codasyl 1971], the network data model includes many schema definition features which appear to be motivated by physical implementation considerations (such as details of the method used to store set types). Moreover, the logical structures in a network schema are often viewed as those to be physically implemented as well. This means for instance that the set types defined in a network schema specify the physical access paths that will be maintained in the implementation. Thus, in the Codasyl model, data independence has been to a large extent compromised. By contrast, it is stressed that a relational schema provides no detail of how t' data will actually be stored [Codd 1970].

2. *The Codasyl DBTG model is very complex.* It contains a very large number of features, whose specific utility in semantic modelling is not particularly clear. On the other hand, the relational data model is quite simple, and its features are very easy to understand. As we have pointed cut, this implies that the relational model suffers from semantic overloading, i.e., that the relation is used to model many different types of semantic information. The network model provides features which correspond to natural constructs for modelling, such as repeating groups (multi-valued attributes), and a specification of which logical links make sense. Unfortunately, the model is cluttered with less useful features, and many of its features are on too low a level (e.g., the concept of a record).

3. *Data selection and modification languages for network systems tend to be oriented to*

record-at-a-time access. "Navigation" [Bachman 1973] from one record to another is stressed. By contrast, most of the data selection and modification languages developed for relational data base management systems are oriented to dealing with sets of things. Furthermore, relational languages tend to isolate the user from implementation detail, while in network systems the user must be conscious of the physical realization of the data base [Chamberlin 1974, Chamberlin 1976c, Held 1975a, Zloof 1975a]. It should be noted that it is certainly possible for a network data model to avoid details of physical storage and access methods, but traditionally this is not the case. Also, it is possible for a high level, nonprocedural, set-oriented retrieval language to be built for a network data base management system [Fehder 1974, Held 1975b] as well as for a relational system.

4. Another important result of the debate concerning which of the conventional data models is most useful, is in fact that none of them is best in all cases. It is important then to provide a data model which incorporates the best aspects of each of these conventional models, while at the same time raising the level of the user's view of a data base.

We have noted that details of physical data storage and access methods tend to be present in the network, and to a large extent the hierarchical, data models. However, if this information is removed from the logical data model, the use of a hierarchy or network to organize a data base has important ramifications on modelling. We will discuss these modelling issues by comparison with the semantic problems of the relational data model described above. In this comparison, we shall deal simultaneously with networks and hierarchies: since hierarchies are a special case of networks, we stress the network model and comment on how the issues differ for the hierarchical model, when relevant.

2.7.1. Semantic Modelling with Hierarchies and Networks

Many of the semantic problems associated with the relational data model are also present in the network and hierarchical data models, however some differences are also important:

1. The problem of "syntactic data structures" and its associated subproblems are present in the network (and hierarchical) models, e.g., the "record type" data structure is representational. However, the DBTG "set type" does provide a better means of modelling interconnections of entities (records). These information-bearing structures are similar in purpose to relational "joins", but they are explicitly declared in the schema. This has the advantage of allowing the data base designer to specify which interconnections make sense.

The hierarchical data model goes a step further towards imposing a strong structure on a data base: a single tree structure is specified for each data base. The important point here is that imposing a hierarchical structure on a data base does serve to semantically structure a schema. For example, placing under a given captain all the ships he has commanded makes it easy and natural to ask about a captain's commands. The problem though is that of determining a single hierarchical structure that can accomodate many different data base uses and views, e.g., allowing a user to find all of the captains that a given ship has had. It is this problem that makes the hierarchical data model unattractive in many cases. (The reader is encouraged to try to design a hierarchical data base for the TMAE.)

2. The network data model suffers from a "name orientation" as does the relational data model. However, the "set type" construct does eliminate the need for some kinds of explicit cross referencing between related records: if a "set type" connect two "record types", a user can travel from the owner to the member records directly. If an appropriate "set type" is not defined, then the user may need to explicitly cross reference between records as in the relational data model. The "set type" also relieves problem of "disjoint structures" to some extent, in that a mechanism is provided for connecting related records. Of course, there is only a single mechanism which must be used for many different semantic types of connections.

3. Since multi-valued attributes are accomodated in the network model (repeating groups), the network model does not exhibit the problem of "single-valued attributes".

4. Some "ordering capabilities" are present in the DBTG model, which allow record and set types to be ordered in limited ways. This is better than in the relational data model, but the facilities are still too limited, e.g., a single ordering must be selected.

5. Network schemas accomodate some types of redundancy which are not commonly present in relational data bases. For example, "automatic" insertion of a new record into a specified set in a data base can be triggered in a network data base; the set into

which this insertion is to be made is determined by other information in the data base. This illustrates a form of redundant/derived information in the network model.

6. The problem of "dynamic subset by name" is handled in a sense in the network data model: the "set type" construct can be used to define a user controllable link between records.

7. The problems of "atomic value sets", "nonintegrated subtypes", "rigid attributes", "aggregation by name association", "artificial abstraction", "static orientation", "inflexible degree of binding", "lack of semantic update units", "superimposed views" and "unexpressed semantic information" are all present in the network and hierarchical data models. The problem of "information in relation and column names" has a counterpart in the network model, namely concerning the naming of "record types" and "set types". The problem of "artificial association entities" is not only present in the network model, but the situation is even worse than in the relational model: since set types only support one-to-many connections, many-to-many associations must be modelled by introducing a dummy relationship record.

In summary, while the network and hierarchical models provide partial solutions to a few of the semantic modelling problems, these models fall far short of the comprehensive semantic data model which is desirable. The modelling degrees of freedom introduced in the network data model are an advantage over the relational model, in that it is possible to model things is a way appropriate to the semantics of the application environment. For example, to model a repeating attribute, the designer has the option of selecting a relational type interfile (interrelation), a repeating group, or an interrecord "set type" connection. This is an advantage, in that one can model things in a way most consistent with the semantics of the application environment. Unfortunately, as in the relational data model, the vocabulary of data structures and interconnections provided for modelling is on a level significantly lower than desirable from the user's point of view: there is a poor correspondence between the level of modelling construct needed and the facilities offered by the DBTG model.

2.8. Extensions and Modifications of the Conventional Data Models

In response to many of the problems discovered in conventional data models, several types of extensions of the relational data model have been proposed. In this section, we briefly examine the proposed extensions by outlining the major thrust of recent work; we stress the problems they do and do not solve:

1. The relational data model has been extended to include capabilities for supporting multiple user views (as discussed above).

2. The direct extension of the relational data model to include "semantic integrity constraints" of various types has been proposed [Eswaran 1975, Hammer 1975a, Hammer 1976d, McLeod 1976b, McLeod 1976e, Stonebraker 1974b]. This approach basically consists of enforcing predicates on states of a data base or transitions between data base states. The semantic integrity constraints are stated as a part of the data base schema. The constraints are predicates that are automatically enforced by the data base

management system, and an appropriate action is taken if a predicate is found to be violated. It is important to note that semantic integrity constraints are superimposed on top of a relational data base, and that no reasonable guidelines have been developed for guiding the design of a complete and consistent set of constraints. We believe that it is better to integrate semantic constraints into the data model itself, viz., into the semantic data structures of the data model.

3. Several recent efforts have focussed upon how semantic information can be captured in the relational data model, and upon how it should (ideally) be captured in that model:

a. Schmid and Swenson [Schmid 1975a] have investigated how semantic information is captured in the relational data model, by classifying relations by "type", e.g., relations which contain autonomous objects, those that describe repeating characteristics of objects, and those which capture associations among objects. This work clarifies the ways in which relations are used to model application environment semantics. However, it does not really attack the issues from the user's viewpoint, and it introduces some distinctions not clearly useful to a user. In sum, this work attempts to explain the uses of relations, but it does not attempt to extend the power of the relational model in capturing semantic information useful to a data base's users.

b. In a somewhat similar way, Wiederhold [Wiederhold 1977] has also analyzed relation types. He distinguishes relations which capture entity existence, relations that match an entity to one of its names ("lexicons"), relations which establish associations among entities, and relations that are used to model multi-valued entity attributes ("nests"). Again, this work explains the ways in which relations can be used, but does not propose guidelines for how they should be used, not does it try to extend the ability of the relational model to capture semantic information.

c. The "hyperrelational model" of Chang [Chang 1975] has attacked the problem of vertical connection, by allowing subrelations to be defined, e.g, defining a set of relations for each convoy by grouping the tuples in CONVOY on common value of Convoy_name and placing them in the relation with the value of Convoy_name as its name. This work is closely related to the work on relational views, and has focused on a specific extension of the relational model.

d. Smith and Smith [Smith 1977a, Smith 1977b, Smith 1977c, Smith 1978a] have presented a design methodology for relational data bases based on the semantics of the application environment. In particular, they propose "generalization" (related to "is a" relationships) and "aggregation" (collecting together attributes of an entity) as concepts in terms of which to structure the relational data base design process. Although their work identifies several important issues, we believe that it suffers from two principal difficulties:

i. The semantic primitives that it provides are too limited.

ii. The work is oriented towards relations, and is couched in terms of them.

We shall have a good deal more to say about this important work in chapter 4.

4. A number of data models have recently surfaced that can be described as "entity - property - association" models:

a. the entity relationship model of Chen [Chen 1976, Chen 1977a],

b. the work of McGee [McGee 1976],

c. and the entity - property - association model of Pirotte [Pirotte 1976, Pirotte 1977].

Chen's entity relationship model is representative of these three models, and we briefly describe it here. In this data model, information is organized into entity sets and relationship sets. An entity set is a collection of entities of a given type (e.g., a ship), and a relationship set is a collection of associations among entities that are of the same kind (e.g., assignments of captains to ships). The entities involved in an association play specified roles therein, e.g., captain and ship in the relationship set associating ships with their current captain. The roles in an association can be constrained to satisfy functionality constraints, e.g., each ship has at most one captain. Each entity and relationship set has an associated collection of attributes, which are the characteristics of that entity or relationship type. The values of attributes are taken from specified "value sets", which are collections of atomic data objects (e.g., person name, date, etc.).

Entity and relationship sets are represented in table form: entity relations contain one tuple for each entity, and relationship relations contain one tuple for each instance of an association. A unique identifier ("primary key") for each such relation is selected. In some cases the tuples of an entity or relationship relation cannot be uniquely identified by the values of their own identifiers; in this case the participation of an entity or association in some relationship is used to identify it (a "weak" entity or relationship relation). Distinguishing weak entity and relationship sets allows a certain type of integrity constraint to be expressed, i.e., deleting a relationship relation tuple causes automatic deletion of weak entities in the relationship.

The principal problem with the entity relationship data model, as well as the other "entity - property - association" models are:

a. The structures of the model are too closely tied to relations. Notably, these data models have been mainly described as extensions of the relational data model and design aids for guiding the construction of relational data bases. (Chen has recently examined the use of the entity relationship model to design network data bases as well [Chen 1978a].) We believe that even a well designed relational data base cannot naturally and effectively model a complex application domain.

b. The proposed models are also insufficiently rich and do not offer the range of flexible modelling capabilities desirable. For example, the entity relationship model: does not directly accomodate associations among relations [Chen 1976]; does not allow entities to be used as values of attributes (but rather forces the use of atomic "value sets"); does not capture essential semantic differences, such as the distinction of objects and event.

In sum, although these research efforts have contributed valuable insights into the problem of semantic data modelling, none of them has taken an integrated approach to the whole problem.

2.9. Other Related Approaches

There have been a number of recent studies in the area of data base semantics that have appeared outside the context of the relational data model. Here, we summarize the highlights, relevant implications, and shortcomings of this work:

1. Several functional dependency based approaches have been suggested, wherein the design of a data base is based directly upon functional dependencies between values in it. Thus, the fundamental data structuring concept is that of relating sets of values by means of the functional relationships between them. Examples of this approach are:

   a. the functional model [Kerschberg 1976d, Sibley 1977a],

   b. the synthesis approach to designing relations from a set of functional dependencies [Bernstein 1975a, Bernstein 1975b, Bernstein 1975c, Bernstein 1976],

   c. the decomposition approach to generating a "good" set of relations from a "bad" one, by using functional dependencies [Codd 1971b, Codd 1971c],

   d. extensions to multi-valued dependencies [Fagin 1977b, Paolini 1977b, Pelagatti 1977, Sharman 1976].

   The principal shortcoming of this approach is that functional dependencies are adequate to capture only limited types of semantic information.

2. Several data models that present a binary relational logical schema to users have been proposed, most notably:

   a. the entity set model [Senko 1974, Senko 1976a, Senko 1976c, Senko 1977], which is based on nonredundant, symmetric binary associations between entity sets and entity name sets (in the DIAM II infological level),

b. the binary relational model [Bracchi 1972, Bracchi 1974, Bracchi 1976],

c. the data semantics model [Abrial 1974].

One of the main problems with binary relational data models is that it is difficult to easily accomodate n-ary relations: (potentially artificial) association entities must be introduced. Another problem with the three approaches described above is that, as is the case for the "entity - property - association" models, insufficient richness of modelling constructs is provided.

3. The "link and selector language" [Tsichritzis 1976a] has provided a unification of many of the advantages of the relational, network, and hierarchical data models vis-a-vis logical data modelling. Here, record types are flat files (relations) with duplicates and an ordering, and links are explicit joins (and do not carry information). Selectors are used to define restrictions of record types by means of a predicate on the fields of the record. The contribution of this data model vis-a-vis data modelling is that it has revealed some of the commonalities of the relational and network data models, e.g., by supporting explicit specification of which logical interrecord connections make semantic sense.

4. The "role model" [Bachman 1977b] has been developed as an extension of the network data model. Here, the roles that entities can play are emphasized, e.g., persons can play the role of captains of ships, ship inspectors, etc. This work highlights the importance

of distinguishing the specific semantic relationships that can exist between entities.

Each of these efforts has made important and specific observations concerning the relevance of including certain types of semantic information in a data model. However, none of these efforts has attempted to provide a comprehensive solution.

2.10. Knowledge Representation in Artificial Intelligence

Researchers in the field of artificial intelligence (AI) have long been concerned with the problem of "knowledge representation", i.e., with schemes for storing and accessing knowledge in computers. The importance of the need for effective modelling capabilities to facilitate the construction of computerized data bases that closely mirror the inherent semantics of an application environment has been recognized by AI researchers. The work on "semantic networks" exemplifies the concern in artificial intelligence research with problems of data base management, such as the modelling of entities and the interconnections among them [Bobrow 1977a, Bobrow 1977b, Fikes 1977, Hawkinson 1975, Hayes 1977, Hendrix 1977b, Roussopoulos 1975, Szolovits 1977].

At the highest level, much of the work in AI and data base management have related goals, namely that of usefully structuring large quantities of information. However, there are a number of ways in which our approach differs from the work in AI knowledge representation:

1. Although we are not settling for a very formal, extremely simple data model (like the relational data model), we are not attempting to solve the complete problem of

knowledge representation. That is, we are not attempting to provide a framework to facilitate the expression of a wide variety of "general" and "common sense" knowledge. Rather, we are focusing on capturing in a data base description the types of semantic information useful in typical data base applications.

2. We are oriented toward providing a framework for data modelling, rather than establishing a basis for the modelling of "intelligent" processes. For example, we are concerned with providing a data base structure that relates the structure that models "oil tankers" to the structure that models "ships" (i.e., that oil tankers are a special kind of ship), rather than emphasizing that fact that "oil tanker" is a concept that is a specialization of the concept of "ship".

3. The embedding of information *concerning the solution to* application problems is not an explicit concern in our work. We are interested in capturing data and its meaning, but are not attempting to develop capabilities for actually accomplishing problem solving activities in the application environment (such as scheduling actions, automatically running factory machinery, etc.)

2.11. Conclusions

In the above analysis of semantic problems with conventional data models, we have specifically focused on the relational data model, pointing out its many problems with respect to semantic data modelling from the viewpoint of the data base user. We do not intend to belittle the importance and utility of many of the ideas introduced by recent work on relational data base management techniques. Rather, we suggest that a new research direction needs to be fruitfully followed: to provide a higher level semantic data model. Thus we build upon much of the work on relational and other conventional data modelling techniques.

Previous work in data base management which has directly or indirectly focussed on semantic data modelling has revealed a number of important issues and has proposed useful modelling techniques. Similar contributions have been made by the network and hierarchical data models. We draw on this work in our approach. These efforts have mainly focused on one or two insights into semantic modelling; these efforts for the most part have not provided a comprehensive and integrated approach to semantic modelling. In the next chapter we describe a new data model (data base structuring mechanism) which attempts to provide such an integrated approach.

3. THE SEMANTIC DATA MODEL

This chapter describes the SDM (semantic data model) in detail. The emphasis in this chapter is on providing a specification of the SDM; in chapter 4, the SDM is discussed, as well as the effectiveness of the SDM in modelling natural structures in an application environment.

3.1. The Basic Structure of an SDM Data Base

The SDM is designed to provide features for the natural modelling of data base application environments. To design the SDM, we analyzed many data base applications, assessed the shortcomings of conventional data models in capturing the semantics of these applications, and designed strategies to handle the important problems revealed. The design process was iterative, in that features were removed, added, and modified during various stages of design.

The process of designing the SDM has provided a detailed data model which is specified in this chapter. In designing the SDM, we have been guided by the following general principles of data base organization:

1. A data base should be and is viewed as a collection of *entities*, which are the relevant abstract objects in the application environment.

2. The entities in a data base are organized into *classes*, which are meaningful collections of relevant abstract objects.

3. The classes of a data base are not in general independent, but rather are logically

linked via *interclass connections.*

4. Data base entities and classes have *attributes*, which describe their characteristics, and relate them to other data base entities.

5. A spectrum of ways of defining interclass connections and attributes is provided, corresponding to the common types of constructs appearing in data base applications. A structured stepwise mechanism is provided to define more complicated kinds of attributes and interclass relationships.

3.2. Classes and Entities

An *SDM data base* is a collection of classes. The structure and organization of an SDM data base is specified in an *SDM schema*, which specifies the classes in the data base. Figure 3-1 contains an example SDM schema for a portion of the "tanker monitoring application environment", defining a number of classes. Examples in this chapter are based on this application domain, which is concerned with monitoring and controlling ships with potentially hazardous cargoes, e.g., oil tankers, as they enter U.S. coastal waters and ports. A data base supporting this application would be used to keep track of ships, oil tankers, ship names, tanker inspections, oil spills, ships that are banned from U.S. waters, and so forth.

Each class in an SDM schema has the following features:

1. A *class name* identifies the class. (For notational convenience, class names are specified by strings of upper case letters, which may be connected by the character "_".)

Each class in an SDM schema has a distinct name.

2. Each class has a collection of *members,* which are the entities that belong to it. The phrases "the members of a class" and "the entities in a class" are thus synonymous. Each class is a homogeneous collection of one of the following types of members (*entity types*):

a. *Objects* are concrete or abstract entities in the application environment. There are the following types of objects:

i. A *concrete object* is an entity (either physical or intangible), such as a ship or a port. In figure 3-1, classes SHIPS, OFFICERS, COUNTRIES, and OIL_TANKERS are examples of concrete object classes. (Note that we do not mean to ascribe any strong philosophical interpretation to the terms "concrete" and "abstract" here.)

ii. An *abstraction* is a generalization of another entity; loosely, the purpose of an abstraction is to "abstract the essence" of another kind of entity. As such, abstractions are related to the notions of "generalization" and "block" as described by Smith and Smith [Smith 1977a, Smith 1977b, Smith 1978a]. An example of an abstraction class is SHIP_TYPES (as shown on one of the latter pages of figure 3-1).

iii. An *aggregate* is a (homogeneous) collection of another type of entity. For example, a ship convoy is an aggregate of ships (see class CONVOYS in figure 3-1). (Note that the term "aggregate" is used here in a different way from the way it is used in the related work of Smith and Smith [Smith 1977a, Smith 1977b, Smith 1978a].)

b. *Events* are actions or activities in the application environment in which objects participate:

i. A *point event* is an event whose essential feature is its occurrence, such as a ship accident (classes INCIDENTS and OIL_SPILLS).

ii. A *duration event* is one whose duration is of interest, e.g., the assignment of a captain to a ship (class ASSIGNMENTS).

c. *Names* are designators for objects or events. A name is used to stand for and thereby replace an object or an event when the nature and structure of that object or event are not of interest. A name is representational, in that it is a number or a string, rather than some actual entity in the application environment. The name classes STRINGS and NUMBERS are built into the SDM, and are used to construct other name classes. Examples of defined name classes are SHIP_NAMES (the set of all possible names of ships), DATES (calendar dates), and KNOTS (speeds in knots).

Figure 3-2 contains a summary of the possible types of members an SDM class may contain.

3. Each class has a collection of attributes which describe the members of that class or the class as a whole. There are three types of attributes, classified according to their *applicability*:

a. A *member attribute* describes an aspect of each member of a class, by logically connecting the member to one or more related entities in the same or another class. Thus, a member attribute is used to describe each member of some class. For example, each member of class SHIPS has attributes Name, Captain, and Engines, which link a ship to its name, current captain, and engines (respectively). The Name and Captain attributes are defined in figure 3-1 as member properties, while Engines is a component member attribute (i.e., it models a component of a concrete object). (A more detailed discussion of attributes is provided in section 3.3.)

b. An attribute of each member of a class that has the same value for all members of that class is a *class-determined attribute*. Such an attribute is a member attribute, but it is associated with the class as a whole because the attribute has the same value for all class members. For example, to capture the fact that no oil tanker can sail faster than some top speed, the class-determined attribute Absolute_top_speed of class OIL_TANKERS is defined (see figure 3-1).

c. A *class attribute* describes a property of a class taken as a whole. For example, the class INSPECTIONS has the attribute Number, which gives the number of inspections currently in the class.

4. Each class may (optionally) have a *class description*, which is a textual explanation of the nature, purpose, and uses of the class. This is a documentation aid. Class SHIPS in figure 3-1 has such a description.

5. Each class is either a *base class* or a *nonbase class*. A base class is one that is defined independently of other classes in the data base, while a nonbase class is defined in terms

of one or more other classes. (Nonbase classes are discussed in detail in section 3.2.1.)

6. For a base class, the *class member kind* is explicitly specified as one of the following:

   a. a concrete object class (e.g., SHIPS),

   b. a point event class (e.g., INCIDENTS),

   c. a duration event class (e.g., ASSIGNMENTS).

7. Each base class is specified as either *containing duplicates* or *not containing duplicates*. (The default is no duplicates.) A class with duplicates models a multiset/bag of entities; this means that some of the members of the class are indistinguishable. For example, a multiset class can be used to model a collection of spare ship engine parts of a given type, since they are equivalent for the purposes of repair.

8. Each base class has an associated list of groups of member attributes; each of these groups serves as a key to uniquely identify the members of a class (*identifiers*). There must be a one-to-one correspondence between the values of each identifying attribute or attribute group and the entities in a class. For example, class SHIPS has the unique identifier Name, as well as the (alternative) unique identifier Hull_number. Of course, multiset classes have no unique identifiers.

9. For a nonbase class, an interclass connection is specified, whose purpose is to define the class in terms of others. In the schema definition syntax (as in the example in figure 3-1), the existence of a *derivation* for a class means that it is nonbase; the derivation specifies which interclass connection is used to related the class to other class(es) in the data base. Each nonbase class thus has associated with it exactly one interclass

connection.

3.2.1. Interclass Connections

Figure 3-3 contains a few examples of SDM classes and their interclass connections. The meaning of these (and several other) interclass connections is explained in this section. In all, there are seven types of interclass connections in the SDM. These seven interclass connections are organized into two main categories:

1. The first five interclass connections allow a nonbase class (C) to be defined whose members are of the same basic entity type as those in the class(es) to which C is related (via the interclass connection). This type of interclass connection is used to define a subclass or a superclass of a given class. A *subclass* S of a class C is a class that contains some, but not necessarily all, of the members of C. A *superclass* S of a class C is a class that contains at least all of the members of C. Note that the very same entity can thus be a member of many classes, e.g., a given ship entity may be a member of classes SHIPS, OIL_TANKERS, and MERCHANT_SHIPS. This is distinctly different from most data models, e.g., the relational data model, wherein a tuple is a member of exactly one relation.

The five interclass connections in this first category are as follows:

a. *Restrict* defines a subclass in terms of attribute(s) of a *parent class*. A restriction subclass C2 contains precisely those members of Cl which satisfy some specified predicate on the members of Cl. The very same entities that are members of the subclass C2 (defined by restriction on Cl) are also members of the parent class Cl. A restriction of a class of concrete objects is obviously a class of concrete objects; analogous statements are true for classes of (point and duration) events, and names. MERCHANT_SHIPS is an example of a class defined by restriction, as shown in figure 3-1; the restriction predicate here on members of SHIPS is "Type = 'merchant'", which requires the attribute Type to have the value "merchant". Similarly, class OIL_SPILLS is defined as a restriction of class INCIDENTS.

b. *Subset* defines a user-controllable subclass of another. A subset class C2 contains at all times only entities that are members of Cl. However, unlike restriction, the definition of C2 does not identify which members of Cl are in C2; rather, data base users *explicitly* add to (and delete from) C2, so long as the subset limitation is observed. For example, BANNED_SHIPS is defined as a subset of SHIPS, allowing some authority to ban a ship from U.S. waters (and possibly later remand that ban).

An essential difference between restriction and subset subclasses is that the members of a restriction are determined by other information in the data base, while a subset's members are directly and explicitly identified as such by users. A restriction defines a permanent subclass of the parent class, in the sense that the predicate which defines it is fixed. The members of a restriction can of course change over time, but only when attribute values for members of the parent class are modified (e.g., MERCHANT_SHIPS will gain a new ship as a member if the value of attribute Type is set to "merchant" for some members of SHIPS.) By contrast, subclasses defined as subsets are more loosely defined: users routinely add members of the parent class to a subset (and may subsequently remove them).

c. *Merge members* allows a new class (C) to be defined whose members are precisely those that exist in either of two *underlying classes* (U1, U2). If the underlying classes are sets (have no duplicates), then the "merge members" interclass connection is equivalent to a set union. (A complete discussion of how duplicates are handled is provided in section 3.3.4.) For example, figure 3-1 contains the class SHIPS_TO_BE_MONITORED, which has a derivation that defines it as the merge members of BANNED_SHIPS and OIL_TANKERS_REQUIRING_INSPECTION. Figure 3-4a illustrates the definition of SHIPS_TO_BE_MONITORED.

It is required that the underlying classes U1 and U2 have a common *root class* (R) in the SDM schema; that is, U1 and U2 must be eventual subclasses of some SDM class R. (Multiple levels of subclasses leading to R are acceptable.) In the example, BANNED_SHIPS is defined as a subset of SHIPS and OIL_TANKERS_REQUIRING_INSPECTION is a subset of OIL_TANKERS (which is in turn a restriction of SHIPS); thus, U1 and U2 have a common root class (namely, SHIPS). If duplicates are present in one or both of the underlying classes, "merge members" places multiple occurrences in C equal in number to the maximum of the number of occurrences in U1 and the number of occurrences in U2.

In addition to "merge members", the SDM provides two other *multiset operator* *interclass connections*: namely those corresponding to set intersection and set difference. These three interclass connections are included in the SDM because it is often natural for a user to think of a class as being related to others as an intersection, union, or difference. These interclass connections are also necessary for completeness: since user-controllable subsets are present in the SDM, the restriction mechanism is not sufficient to permit the definition of certain types of subclasses involving subsets. For example, class intersection rather than restriction must be used to define class SHIPS_TO_BE_MONITORED, since BANNED_SHIPS and OIL_TANKERS_REQUIRING_INSPECTION are subsets: there are no relevant member attributes of either of these classes which can be used to state an appropriate restriction predicate.

d. *Extract common members* allows two SDM classes to be intersected. For example, the class BANNED_OIL_TANKERS is defined via "extract common members" applied to classes OIL_TANKERS and BANNED_SHIPS (see figure 3-4b). As for "merge members", the underlying classes (U1 and U2) must have a common root class. And, if duplicates are present in one or both of the underlying classes, "extract common members" places multiple occurrences in the new class C equal in number to the minimum number of the times that member occurs in U1 and U2.

e. *Extract missing members* has the effect of defining a new class that is the set difference of two SDM classes (U1 and U2). Figure 3-4c illustrates the definition of class SAFE_SHIPS, as consisting of those members of SHIPS not in the class

BANNED_SHIPS. Again, the underlying classes must have a common root class. Also, if duplicates are present in one or both of the underlying classes, "extract missing members" places multiple occurrences in the new class (C) equal in number to the difference between the number of times that the member occurs in U1 and the number of times it occurs in U2; if the member occurs more times in U2 than in U1, it is not a member of C.

2. The remaining two interclass connections allow a nonbase class (C) to be defined whose members are of a different basic entity type than those in the underlying class (U). These two interclass connections allow higher level, abstract entities to be constructed out of basic entities. In particular, the interclass connections that allow such *second order classes* to be defined are:

a. *Abstract* defines a class whose members are generalizations of the members of another class. The members of an abstraction class are called *abstractions*. For example, the abstraction class SHIP_TYPES is defined by the interclass connection "abstract" on class SHIPS (see figures 3-1 and 3-3). The members of SHIP_TYPES are types of ships: merchant, fishing, and military (in the TMAE data base). Ships themselves are not the members of SHIP_TYPES, but are rather *instances* of each type (member of SHIP_TYPES).

To define an abstraction class, a grouping expression on the class underlying the abstraction must be provided. This grouping expression collects the members of the underlying class into classes that contain the instances of the corresponding abstractions. The grouping allowed in an abstraction interclass connection is on common value of one or more member attributes of the underlying class. For example, the grouping expression in the definition of class SHIP_TYPES in figure 3-1 is "on common value of Type" (in SHIPS). Note that if the grouping expression involves only a single-valued attribute, then the instances classes partition the class underlying the abstraction (as for SHIP_TYPES); if a multi-valued attribute is involved, then the instances classes may overlap.

The instances of a member of an abstraction class are a restriction of the class underlying the abstraction. The grouping expression used in the specification of the interclass connection "abstract" thus corresponds to a collection of restriction expressions. For example, for SHIP_TYPES, the grouping expression "on common value of Type" corresponds to the restriction predicates (on SHIPS) "Type = 'merchant'", "Type = 'fishing'", and "Type = 'military'". In fact, some or all of these restrictions may be independently and explicitly defined in the data base. In figure 3-1, the class MERCHANT_SHIPS is defined as a restriction of SHIPS, and it is also listed in the definition of SHIP_TYPES as an instances class that is explicitly defined in the data base. In general, when an abstraction class is defined, a list of the names of the instances classes that are explicitly defined in the data base should be included in the specification of the interclass connection; the purpose of this list is to highlight independent uses of restriction predicates that are instances of the grouping expression.

b. *Aggregate* defines a class whose members are *aggregates*. An aggregate is a collection of entities of a given type. The entities that constitute an aggregate are called its *constituents*, and form a subset of the members of the class underlying the aggregation. For example, the class CONVOYS is defined in figure 3-1 as a (primitive) aggregate of SHIPS; thus, each member of CONVOYS is a collection of SHIPS (i.e., the constituents of a convoy are ships).

The constituents of a member of an aggregate class are a subset of the class underlying the aggregation. This is analogous to viewing the instances of a member of an abstraction class as a restriction of the class underlying the abstraction. An aggregate differs from an abstraction much as a subset differs from a restriction. The essential difference is that the definition of an abstraction class determines the instances of each member (abstraction), while the definition of a (primitive) aggregate class does not determine the constituents of each aggregate.

All, some, or none of the subset classes which contain the constituents of an aggregate may be separately and explicitly defined in the data base. A *primitive aggregate class* is one for which none of the members of the aggregate class has an existence as an independent subset class, separately defined in the schema. For example, it may well be the case that the class CONVOYS is to exist as an aggregate of SHIPS, but the individual convoys themselves are not separately defined as data base classes (subsets of SHIPS). This type of aggregate class is defined by the interclass connection *primitive aggregate*, which requires only the specification of the class underlying the aggregate (e.g., class CONVOYS in figure 3-1). Alternatively, a *derived aggregate class* is one for which all of the subsets that correspond to the constituents of the members of the aggregate class are explicitly defined in the data base. For example, the classes CONVOY_05301, CONVOY_43411, and CONVOY_42343 might be independently and explicitly defined as subsets of class SHIPS. Here, the interclass connection *derived aggregate* is used, and the names of the constituents classes are listed in the definition of the aggregate class. A *mixed aggregate class* is one for which some of the constituents classes are explicitly defined in the data base and some are not. The interclass connection *mixed aggregate* requires that the names of the constituents classes defined in the data base be listed, and that the underlying class from which the remaining aggregates are to be formed be specified. Observe that in the case of a derived or mixed aggregate, the aggregate definition declares relationships between independently defined classes that would otherwise be hidden.

A summary of the interclass connections in the SDM is provided in figure 3-5.

3.2.2. Name Classes

Entities are abstract concepts which are directly modelled in an SDM schema. In the real world, abstract entities can be denoted in a number of ways, e.g.,a ship can be referenced by its name, its hull number, a picture of it, or by pointing one's finger at the ship itself. Since abstract ship entities are captured in an SDM data base, the SDM user needs some way of referring to these internal entities. In order that it be possible for information in a data base to be communicated to the outside world (users), the data must at some point be entered and displayed on existing computer terminals. This is achieved in the SDM via "names".

As stated above, a name class is a collection of numbers or strings. A class is defined as a name class when its members are best thought of as numbers or strings, and when the details of the entities it models are not of interest. By "name", we mean anything that is a string or a number. We do not insist that names are thought of as proper identifiers of objects or events; in the SDM, dates, colors, and last names of persons may all be considered names.

Aside from the base name classes STRINGS and NUMBERS, which are built into the SDM, application environment specific name classes can be defined. These user-defined name classes are similar to the user-defined domains proposed for the relational data model [McLeod 1977a], in that they are abstract sets of atomic data values. An SDM name class is defined as a subclass of either STRINGS or NUMBERS. Specifically, a name class is defined using one of the following interclass connections: "restrict", "subset", "merge members", "extract common members", and "extract missing members". All name classes are thus ultimately defined as subclasses of STRINGS or NUMBERS; a name class can be defined as a subclass of a name class which is itself defined as a subclass of STRINGS or NUMBERS, etc. By definition, name classes do not have duplicates.

To use "restrict" to define a subclass, one specifies a name class restriction predicate, which allows a pattern to be specified that a member of the parent class must satisfy in order for it to be a member of the class being defined. Figure 3-6 contains two examples of name class definitions using this approach. Classes ENGINE_SERIAL_NUMBERS and DATES are shown in figure 3-6, and are defined as restrictions of NUMBERS and STRINGS, respectively. In essence, the restrict predicate provides a "format" which describes a subclass of STRINGS or NUMBERS. The "ordering" specification in the definition of class DATES means that the data values are ordered by the sub-units named (e.g., so that the ">", ">=", "<", and "<=" operations work correctly).

Note that a "format" specification is strictly local, in the sense that there are no references therein to other information in the data base. Name classes are intended to be underlying sets of names, rather than being derived in some complicated way from the data base. Note for example, that this rules out defining the name class NAMES_OF_US_BASED_SHIPS as the restriction of SHIPS where Home_port satisfies the appropriate predicate. In the example, the names of all ships based in the U.S. would be obtained by extracting the value of the name attribute for each ship based in the U.S.

In the example in figure 3-6, "subset" is used to define a name class which can be readily enumerated. For instance, COUNTRY_NAMES is a class whose members are enumerated, e.g., by typing them into the data base. The members of a subset name class can be dynamically altered by users.

Further details of name class definition are presented in [McLeod 1977a].

3.3. Attributes

As indicated above, each class has a collection of attributes associated with it. Each attribute has the following aspects:

1. An *attribute name* identifies the attribute. (For notational convenience, attribute names appear as one upper case letter followed by lower case letters and the character "_", e.g., Home_port for class SHIPS in figure 3-1.) Attribute names are unique within a class.

2. Attributes may be hierarchically organized. For example, the members of the class SHIPS have the attribute Size, which in turn consists of attributes Length and Draft. The purpose of this hierarchic scheme is to allow related attributes to be grouped together; this is strictly a notational convenience to aid in the organization of the attributes of a class.

3. Each attribute has a *value*. The value of an attribute is either an entity in the data base (a member of some data base class), or a class of such entities. The value of an attribute can also be *unknown*; this special type of value is treated like any other with respect to restrictions and abstractions, e.g., a member of class SHIPS with a Cargo_type "unknown" is not a member of the restriction class OIL_TANKERS.

4. Each attribute has a specified semantic *kind*, which identifies the type of relationship the value of the attribute has with the entity. Each member attribute is identified as being of one of the following kinds:

   a. A member attribute which is not of a more specific semantic kind (as discussed immediately below) is called a (member) *property*. For example, class SHIPS has the member properties Hull_number, Country_of_registry, and Captain.

b. A member attribute of a concrete object class that models a constituent part of that object is called a *component*. Only concrete objects can have component attributes. For example, SHIPS has the component member attribute Engines.

c. A member attribute which models an entity which plays a role [Bachman 1977b] in an event is called a *participant* of the event entity. Only event classes can have member participant attributes. For example, class INCIDENTS has the participant attribute Involved_ship.

d. A class-determined attribute is similarly specified as a *class-determined property*, a *class-determined component*, or *class-determined participant*. These kinds are similar in meaning to the corresponding kinds of member attributes, except that the attribute has the same value for each member of the class. Figure 3-1 shows the class-determined property Absolute_top_speed of OIL_TANKERS.

e. A class attribute that models some aspect of a class taken as a whole is called a *class property*, e.g., class property Number_of_ports of class PORTS (in figure 3-1).

These distinctions of attribute types are useful in capturing the nature of the relationship between a class member and the value of its attributes.

A more extensive vocabulary of semantic attribute kinds could be provided, e.g., by distinguishing the source of information about an event, the occurrence time of a point event, the duration of a duration event, etc. However, such a case grammar analysis [Schank 1973] conflicts with the SDM goal of attempting to support important semantic distinctions while avoiding undue complexity. A large number of special features would be required in such an approach, and their adequacy would be difficult to verify in any event.

5. An *attribute description* is optionally provided, which explains in textual form the nature, role, and use of the attribute.

6. Each attribute is either *primitive* or *derived*. A primitive attribute is one whose value is supplied by a user. A derived attribute is one whose value is calculated from other information in the data base.

7. Each primitive attribute has a specified underlying *value class*, which is the class from which the value of the attribute must be selected. Any class in the schema is a candidate for the value class of an attribute; for example, the value class of member attribute (property) Captain of SHIPS is the concrete object class OFFICERS.

8. Each primitive attribute is either *single-valued* or *multi-valued*. The value of a single-valued attribute is a member of the value class of the attribute, while multi-valued attributes have a value which is a subclass of it. Thus a multi-valued attribute is itself a class, i.e., a collection of entities. In figure 3-1, the class OIL_TANKERS has the single-valued member property Hull_type, the multi-valued member property Inspections, and the single-valued class-determined property Absolute_top_speed.

9. A primitive attribute is specified as *mandatory* if it cannot have an unknown value. Any attribute can be declared mandatory. Unique identifiers are automatically

mandatory, and event participants are normally declared as mandatory.

10. Each primitive attribute has an associated *degree of binding*, which is either "fixed" or "changeable". Fixed attributes have values which do not change during the lifetime of an entity or class, except to correct a data recording error.

11. Each derived attribute has a *derivation specification*, which describes how the value of the attribute is calculated from other data in the data base.

3.3.1. Attribute Derivation Specifications

The SDM provides a vocabulary of semantic types of attribute derivation specifications, which intends to capture the most common types of derived information. These special cases are the *attribute derivation types*. The vocabulary of derivation types is important: that each type performs a different type of meaningful data manipulation, resulting in a value for a derived attribute. A stepwise approach is taken to the definition of derived attributes that are more complicated than those directly accommodated in the vocabulary. That is, complex derived attributes are defined by:

1. describing related attributes,

2. applying one of the types of manipulations accomodated in the vocabulary to yield the desired attribute.

This procedure can be repeated for the related attributes, so that an arbitrarily complex attribute derivation specification can be developed.

There are three main categories of attribute derivation types which allow a derived attribute to be defined for a class (C) (these are detailed in sections 3.3.1.2, 3.3.1.3, and 3.3.1.4.):

1. A member attribute is defined by a *member-specific derivation*. This category of derivation types allows various kinds of derived member attributes to be defined which specify the value of a derived attribute of C in terms of the relationships of a member of C with other information in the data base.

2. A class attribute is defined by a *class-specific derivation*; this kind of derivation is used to describe a derived property of C taken as a whole by specifying its value in terms of aspects of the members of C.

3. An attribute of C is defined in terms of the values of other attributes of C. This kind of derivation type is called an *inter-attribute derivation*. There are three specific possibilities: a member attribute is defined in terms of other member attributes, a class-determined attribute is defined in terms of other class-determined attributes, and a class attribute is defined in terms of other class attributes. As we shall see, the mixing of attributes by applicability is not allowed in inter-attribute derivation (e.g., combining the values of a member attribute and a class attribute); the meaningful manipulations which involve attributes of different applicability are handled by class derivations.

Note that there are no class-determined-specific derivation types; as should become clear later, this is because there are no meaningful manipulations that yield derived class-determined attributes, other than those allowed by inter-attribute derivations.

3.3.1.1. Mappings

Before discussing the attribute derivation types, it is important to describe the concept of a *mapping*. A mapping is an expression which evaluates to a collection of entities. The purpose of mappings is to allow a user to directly reference the value of the value of an attribute (recursively). For example, suppose that the mapping "Captain.Name" is stated for class SHIPS. The value of this mapping, for each member S of SHIPS, is the value of attribute Name for a member O of OFFICERS, where O is the value of Captain for S. In this example, the attributes Captain of SHIPS and Name of OFFICERS are single-valued. In general however, this need not be the case. For example, consider the mapping for SHIPS "Engines.Serial_number". Attribute Engines is multi-valued, which means that "Engines.Serial_number" may also be multi-valued. This mapping thus evaluates to the serial numbers of the engines of a ship.

An arbitrary number of levels are allowed in a mapping. Thus a mapping is written, in general, as a series of attribute names separated by ".". For example, the mapping for SHIPS "Captain.Superiors.Name" evaluates to the names of all of the superiors of the captain of a ship. A mapping such as this one is multi-valued, since at least one of the steps in the mapping involves a multi-valued attribute. The value of a mapping "X.Y.Z", where X, Y, and Z are multi-valued attributes is the class containing each value of Z which is a value of Y for some value of X. In this case, a "merge of members" is performed at each step in the mapping.

As we shall see, mappings are useful in attribute derivation specifications and restriction predicates. They are particularly necessary because SDM entities stand for themselves as attribute values.

3.3.1.2. Member-Specific Derivations

A member-specific derivation allows an attribute A1 of class C1 to be defined in terms of the relationships of a member M1 of C1 with other information in the data base:

1. A1 is defined as an ordering attribute. In this case, A1 is a derived member property formed by assigning to each member M1 of C1 a value for A1 that denotes the sequential position of M1 in C1 when ordered by one or more other specified (single-valued) member attributes (or mappings) of C1. Ordering is by decreasing or increasing value. For example, the attribute Seniority of OFFICERS is defined as "order by Date_commissioned" (decreasing order, by default). Ordering within groups is also possible: "order by A2 within A3" specifies that A1 is to be equal to the sequential position of M1 within the group formed by common value of A3, according to the value of A2. For example, in figure 3-1 the attribute Order_for_tanker of INSPECTIONS is defined as "order by decreasing Date within Tanker", which orders the inspections for each tanker. (Mappings are permitted for A2 and A3.)

2. Attribute A1 is specified as an inversion attribute, in which case it is defined by inverting class C2 on attribute A2. The value of A1 for M1 is the class of those members of C2 whose value of A2 is M1. For example, member attribute Ships_registered_here of COUNTRIES has the derivation "invert Country_of_registry of SHIPS".

3. A matching attribute is defined as follows; for each member M1 of C1:

a. A corresponding member M2 of another class C2 is found which has M1 as its value of member attribute A2.

b. The value of member attribute A3 for M2 is used as the value of A1 for M1.

For example, as shown in figure 3-1, attribute Captain of class SHIPS has the derivation "Officer of match to ASSIGNMENTS on Ship". Here, the value of the derived property Captain of each member S of SHIPS is determined by matching S to a member C of ASSIGNMENTS that has S as its value of attribute Ship; the value of Captain for S is the value of Officer of C. The "merge members" form of match is used when it is necessary to match S to multiple members of C2. For example, if a ship could have multiple captains, one would use "merge members in Officer of match to ASSIGNMENTS on Ship". In a sense, a match is a kind of high level "join" [Codd 1970].

4. Attribute A1 is given a boolean value for M1 that is "yes" (true) if M1 is a member of some other specified class C2, and "no" (false) otherwise. For example, OIL_TANKERS has the derived property Is_tanker_banned?, with derivation "if exists in BANNED_SHIPS".

5. The value of attribute A1 is defined as the result of merging the members of recursively tracing the values of some attribute A2. For instance, attribute Superiors of OFFICERS is defined by the derivation "merge members in repeat over Commander"; the value of the attribute includes the immediate commander of the officer, that

commander's superiors, and so on. Note that Commander has OFFICERS as its value class; this must be true for this kind of attribute derivation to make sense.

Two special kinds of derived attributes are automatically defined in the SDM:

1. When an abstraction class is defined, the derived, multi-valued member property *Instances* is automatically defined. The value of this attribute for a member of the abstraction is the class of members (of the class underlying the abstraction) that are instances of that member. For example, each member of the abstraction class SHIP_TYPES has as the value of its attribute Instances the class of all ships of that type.

2. Similar to Instances of an abstraction class, each aggregate class has a *Constituents* member property, the value of which is the class consisting of all the entities (members of the class underlying the aggregation) that constitute the aggregate entity. For example, the value for each member of CONVOYS of attribute Constituents is a class including precisely those members of SHIPS that currently constitute the convoy.

3.3.1.3. Class-Specific Derivations

There are two kinds of attribute derivations which allow attributes of a class taken as a whole to be defined:

1. An attribute is defined whose value is equal to the number of (unique) members in the class it modifies. For example, attribute Number of OIL_TANKERS has the derivation "number of members in this class".

2. An attribute is defined whose value is a function of a numeric member attribute of a class; the functions supported are "maximum", "minimum", "average", and "sum" taken over a member attribute. The computation of the function is made over the members of the class. For example, one might define the class attribute Total_spilled of OIL_SPILLS as "sum of Amount_spilled over members of this class".

3.3.1.4. Inter-Attribute Derivations

Inter-attribute derivations accomodate the common ways in which new attributes are defined in terms of the values of other attributes associated with the same class. The following kinds of inter-attribute derivations are supported:

1. The value of an attribute is specified as equal to the value as some other attribute or mapping. For instance, in figure 3-1, attribute Date_last_examined of OIL_TANKERS has the derivation specification "same as Last_inspection.Date", i.e., the value of Date_last_examined is equal to the value of the member of class INSPECTIONS which is the current value of attribute Last_inspection of OIL_TANKERS)

2. An attribute has a derivation specification which states that the value of that attribute is defined by some arithmetic expression involving the values of other attributes or mappings. The involved attributes must have numeric values, i.e., they must have value classes that are (possibly recursive) subclasses of NUMBERS. The arithmetic operators allowed are addition ("+"), subtraction ("-"), multiplication ("*"), division ("/"), and exponentiation ("!"). For example, attribute

Top_speed_in_miles_per_hour of OIL_TANKERS has the derivation specification "= Absolute_top_speed / 1.1".

3. "Extract common members" is used on mappings of attributes which are multi-valued, to intersect the members constituting their values. For example, the new attribute Related_officers of OFFICERS is defined by the derivation specification "extract common members in Superiors and Subordinates".

4. Similar to case 3, "merge members" is used in a derivation specification.

5. Also, similar to case 3, "extract missing members" is allowed.

6. The operators "maximum", "minimum", "average", "sum" are applied to an attribute name or mapping, that is multi-valued; the value class of the attributes involved must be a (possibly recursive) subclass of NUMBERS. The maximum, minimum, average, or sum is taken over the collection of entities that comprise the current values of the attribute or mapping.

7. An attribute is defined to have its value equal to the number of members in a multi-valued attribute or mapping. For example, attribute Number_of_instances of SHIP_TYPES has the derivation specification "number of members in Instances". (The special way in which the Instances attribute of an abstraction class is discussed below.) "Number of unique members" is used similarly. "Number of members" and "number of unique members" act differently only when duplicates are present in the multi-valued attribute involved.

8. A multi-valued attribute is defined to have a value that is the restriction of the value

of another attribute. For example, attribute Oil-burning_engines of SHIPS is defined via the derivation specification "restrict Engines where Kind_of_engine = 'oil burning'"; the value of attribute Oil-burning_engines consists of the values of multi-valued attribute Engines which are oil burning. Mappings are not allowed in restrictions. Other means are available for this purpose, and mappings in restrictions would introduce unnecessary complexity; if such a restriction is required, an auxiliary attribute is defined using "same as" on the mapping, and the restriction predicate stated in terms of the auxiliary attribute.

3.3.2. Restriction Predicates

As stated earlier, a restriction is specified by providing a predicate on the members of the class underlying the restriction. The types of restriction predicates that are allowed are limited, but provide very flexible facilities when used in combination with derived attributes.

A restriction predicate is a boolean combination of simple predicates, or merely a simple predicate; the operators used to form such a boolean combination are "and", "or", "not", and "()". A simple predicate has one of the following forms:

MAPPING SCALAR_COMPARATOR CONSTANT

MAPPING SCALAR_COMPARATOR MAPPING

MAPPING SET_COMPARATOR CONSTANT

MAPPING SET_COMPARATOR CLASS_NAME

MAPPING SET_COMPARATOR MAPPING

is a value of ATTRIBUTE_NAME of CLASS_NAME

Here, MAPPING is any mapping; SCALAR_COMPARATOR is one of "=", "~=", ">", ">=", "<", and "<="; CONSTANT is a string or number constant; SET_COMPARATOR is one of "is in", "properly is in", "contains", and "properly contains"; CLASS_NAME is the name of some other class defined in the schema; ATTRIBUTE_NAME is the name of an attribute.

For illustration, an example of each of these six forms is provided below along with an indication of its meaning; the first two predicates and the last predicate define restrictions of class OFFICERS, while the third, fourth, and fifth apply to class SHIPS:

Country_of_license = 'Panama'
(officers licensed in Panama)

Commander.Date_commissioned > Date_commissioned
(officers commissioned before their commander)

Cargo_types contains 'oil'
(ships that can carry oil)

Captain is in RISKY_CAPTAINS
(ships whose captain in the class containing officers that are bad risks)

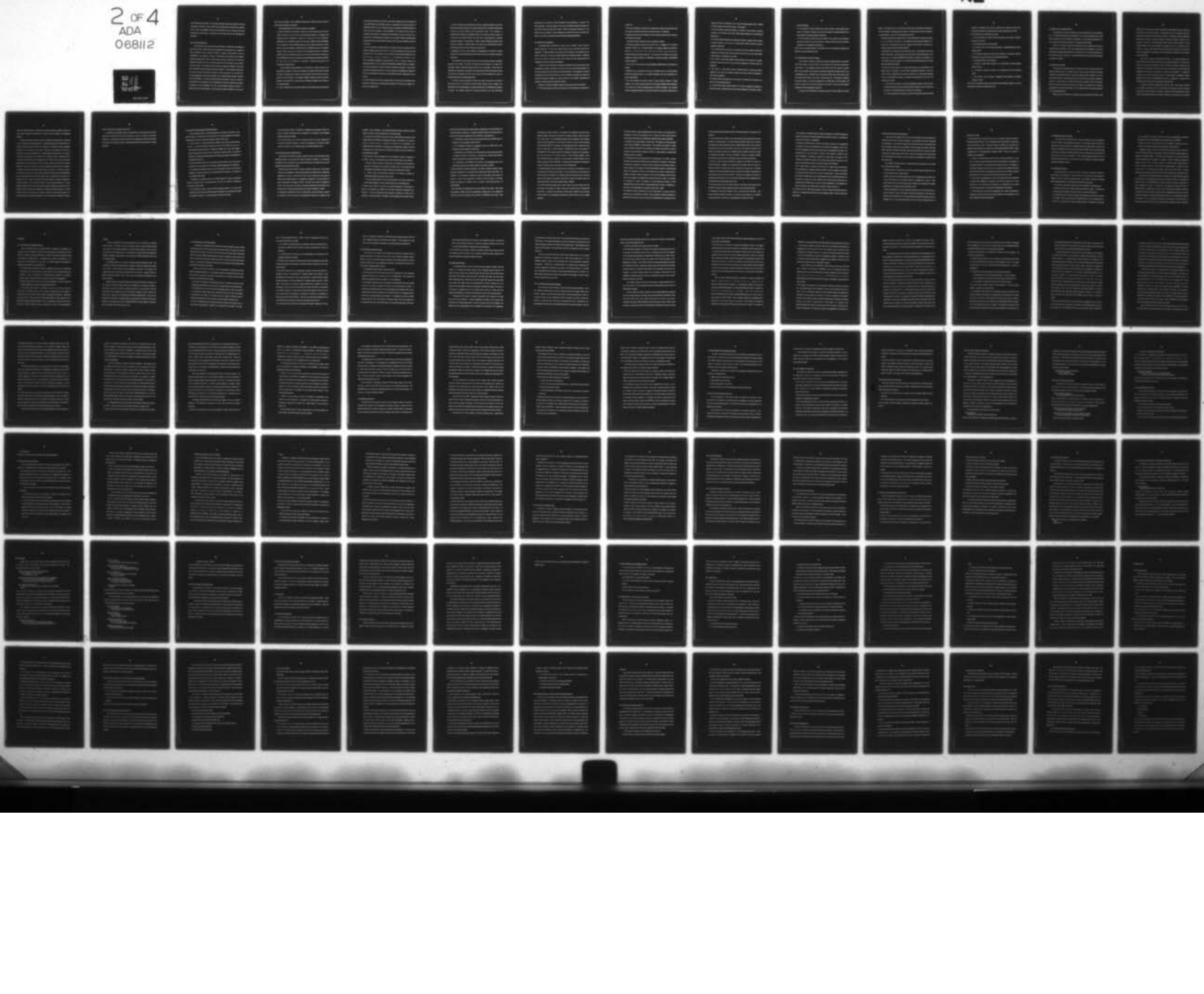
Captain.Country_of_license is in Captain.Superior.Country_of_license
(ships commanded by an officer who has a superior who is licensed in the same country as he)

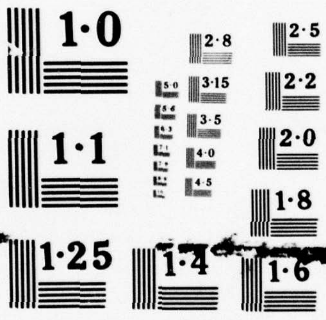
is a value of Involved_captain of INCIDENTS
(officers who were involved in some incident)

Note that the first five predicate types are predicates on the attributes of members of the

AD-A068 112   MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTE--ETC F/G 9/2
A SEMANTIC DATA BASE MODEL AND ITS ASSOCIATED STRUCTURED USER I--ETC(U)
AUG 78 D MCLEOD   N00014-76-C-0944
UNCLASSIFIED   MIT/LCS/TR-214   NL

2 OF 4
ADA
068112




1·0
2·8
2·5
3·15
2·2
5·0
5·6
3·5
1·1
2·0
4·0
4·5
1·8
1·25
1·4
1·6

class underlying the restriction. The last type of predicate means that a member of the class underlying a restriction is to be a member of the restriction class if and only if that member is the value of the specified attribute of the specified class; this is the only restriction predicate type which refers to an attribute of a class other than the one underlying the restriction.

3.3.3. Attribute Inheritance

As noted earlier, it is often the case that some entity in an SDM data base belongs to more than one class. SDM classes can and frequently do share members, e.g., a member of OIL_TANKERS is also a member of SHIPS; a member of OIL_SPILLS is also in INCIDENTS. As a member of a class C, a given entity E has values for each member attribute associated with C. But in addition, when viewed as a member of C, E may have additional attributes which are not directly associated with C, but which are are *inherited* from other classes. For example, since all oil tankers are ships, each member T of the class OIL_TANKERS inherits all member attributes of SHIPS. T has the attributes Hull_type, Is_tanker_banned, Inspections, Number_of_times_inspected, Last_inspection, Last_two_inspections, and Date_last_examined, which are locally defined member attributes of OIL_TANKERS. T also has the attributes Name, Hull_number, Type, etc., which are not mentioned in the definition of OIL_TANKERS, but which are inherited from SHIPS (a superclass of OIL_TANKERS). The value of each inherited attribute of tanker T is simply the value of the attributes of T when viewed as a member of SHIPS; the very same ship entity that belongs to OIL_TANKERS belongs also to SHIPS, so that the value of each inherited attribute is well-defined.

The following specific rules of attribute inheritance are applied:

1. A class C defined as a restriction or subset of a class U inherits from U all of its member attributes as well as all of its class-determined attributes. For example, since LIBERIAN_OIL_TANKERS is defined as a restriction of OIL_TANKERS, LIBERIAN_OIL_TANKERS inherits all of the member attributes of OIL_TANKERS; in turn members of OIL_TANKERS inherit all of the member attributes of SHIPS. Note, of course, that members of LIBERIAN_OIL_TANKERS also have an additional attribute which ships and oil tankers do not (Oil_spills_involved_in). Similarly, LIBERIAN_OIL_TANKERS inherits all of the class-determined attributes of OIL_TANKERS, which in turn inherits all of the class-determined attributes of SHIPS. In this case, two attributes are inherited: Absolute_top_speed and Top_speed_in_miles_per_hour.

Class attributes describe properties of a class taken as a whole, so they are not inherited by a subclass. In order for an attribute to be inherited from class C1 by class C2, both its meaning and its value must be the same for C1 and C2. this is not true in general for class attributes. For example, although a subclass may have a similar attribute to one defined for its parent class, e.g., Number_of_members, their values are not necessarily equal.

2. A class C defined by "extract common members" on the classes U1 and U2 inherits all

of the member attributes of U1 and all of the member attributes of U2. For example, the class BANNED_OIL_TANKERS, defined as containing the common members of BANNED_SHIPS and OIL_TANKERS, inherits all attributes of BANNED_SHIPS as well as all of the attributes of OIL_TANKERS. The reason for this is that, by definition, each member of BANNED_OIL_TANKERS is both an oil tanker and a banned ship. Class-determined attributes are not inherited because class-determined attributes of the same name defined for both U1 and U2 may have different values for U1 and U2.

3. A class C defined by "merge members" on the classes U1 and U2 inherits all of the member attributes shared by U1 and U2, i.e., all of the attributes of U1 that are also attributes of U2. For example, the class SHIPS_TO_BE_MONITORED has the derivation "merge members in BANNED_SHIPS and OIL_TANKERS_REQUIRING_INSPECTION"; thus all attributes shared by BANNED_SHIPS and OIL_TANKERS_REQUIRING_INSPECTION are inherited. In this case, precisely the member attributes of SHIPS are inherited. Only shared attributes are inherited, because C is in effect extracting the commonality of classes U1 and U2. Class-determined attributes are not inherited for the same reasons given in case 2 above.

4. A class C defined as the "extract missing members" in U1 but not in U2 inherits all of the member attributes and class-determined attributes of U1. This case is similar to 1, since C is a subclass of U1.

5. A class C defined as the abstraction of U inherits a member attribute for each class-determined attribute shared by every defined instances class of U (classes containing the instances of each member of the abstraction class). For example, if MERCHANT_SHIPS and FISHING_SHIPS were defined instances classes of abstraction class SHIP_TYPES, and each had a class-determined attribute Absolute_top_speed, C would inherit a member attribute Absolute_top_speed, whose value for a member T of SHIP_TYPES is the value of the class-determined attribute Absolute_top_speed of the class containing the instances of T.

6. Analogous to case 4, a class C defined as the aggregation of U inherits a member attribute for each class-determined attribute shared by all of the defined constituents classes of U.

The inheritance rules described above cover all of the kinds of interclass connections permitted in the SDM. The point of making these rules explicit is to capture the meaningful logical connections between classes via-a-vis attributes. These rules are considered an integral part of the SDM, in the sense that users need not be aware of them. This means, for example, that when a user asks for all of the attributes of a class C, inherited attributes are automatically included.

It would in theory be possible to include other types of rules such as the above inheritance rules. For example, we might include complex rules to support the determination of the relationship of a restriction predicate with the attributes of the subclass it defines. For example, consider the restriction predicate in the class derivation specification of OIL_SPILLS: "restrict INCIDENTS where Description = 'oil spill'". We could formulate a rule that captures the fact that the inherited attribute Description of OIL_SPILLS must have the value "oil spill". However, such predicate analysis is, in the general case, very difficult. Such complexities are beyond the scope of the SDM, and counter to its goals of avoiding the general knowledge representation problem.

3.3.4. Multisets and Duplicates

As specified above, an SDM class is either a set or a multiset: it may or may not contain duplicates. If a class has unique identifiers, then it obviously cannot have duplicates. If unique identifiers are not present, then the default is that duplicates are not allowed. However, a class can be explicitly defined with "duplicates allowed". Duplicates may also be present in attribute values, since attribute derivation specifications and mappings can produce attributes with duplicates.

In point of fact, the existence or nonexistence of duplicates is only of importance when considering the number of members in a class or the size of a multi-valued attribute. On all other occasions, the user need not be concerned whether or not duplicates are present. Consequently, the only SDM primitives that are affected by duplicates are those that concern the number of members in a class, and the size of an attribute. Specifically, the interclass connections deal with duplicates as follows:

1. "Restrict" and "subset" propagate duplicates from the parent class to the subclass.

2. The multiset operator interclass connections deal with duplicates as described in

section 3.2.1.

3. If the class underlying an abstraction has duplicates, the defined instances classes and the values of the Instances attribute of the abstraction class have duplicates.

4. Aggregate classes may have duplicates in their constituents classes and their values of the Constituents attribute.

Attribute derivation specifications deal with duplicates as follows:

1. Mappings propagate duplicates (see section 3.3.1.1.). A mapping "X.Y" contains duplicates if either X or Y contains duplicates. For example, suppose an attribute of SHIPS is defined with the derivation specification "Captain.Superiors.Country_of_license". The value of this attribute will contain duplicates if Captain, Superiors of OFFICERS, or Country_of_License of OFFICERS contains duplicates.

2. The "same as" inter-attribute derivation propagates duplicates from one attribute to another.

3. Duplicates are not permitted to exist for any attribute used in an inter-attribute derivation expression which is an arithmetic expression, since such expressions are always single-valued.

4. The inter-attribute derivations "restrict", "extract common members", "merge members", and "extract missing members" (for multi-valued attributes) handle duplicates in the same way as their corresponding type of interclass connection. For example, "restrict" propogates duplicates from the parent attribute to the subclass attribute (e.g. of

Engines of SHIPS has duplicates, then so does Oil_burning engines, which is defined by "restrict Engines where Kind_of_engine = 'oil burning'").

5. The inter-attribute derivations defining an attribute as the maximum, minimum, average, or sum of another multi-valued attribute include duplicates in the computations they perform.

6. The inter-attribute derivations that compute the number of members and the number of unique members in the value of a multi-valued attribute include and ignore duplicates, respectively.

7. The member-specific derivation which defines a derived member attribute by computing an ordering number, does so based on the value of one or more other single-valued attributes; no duplicates are involved.

8. The value of a multi-valued member attribute defined as the inversion of a member attribute of some other class in the data base has duplicates if that other class has duplicates.

9. The value of a multi-valued member attribute defined by matching contains duplicates if the value of the member attribute of the class to which the matching is done contains duplicates.

10. Existence member attributes have the value "yes" or the value "no", so there are no relevant issues regarding duplicates.

11. A multi-valued member attribute defined as the result of merging the members of recursively tracing a member attribute value contains duplicates if the attribute involved

contains duplicates.

12. A class-specific attribute derivation that is used to define a class attribute whose value is the number of members in the class or the number of unique members in the class, includes and ignores duplicates, respectively.

13. The class-specific attribute derivations defining a class attribute as the maximum, minimum, average, or sum of the value of a member attribute of the class include duplicates in the computations they perform.

These rules for handling duplicates in the SDM cover all of the interclass connections and attribute derivation specification types.

3.4. The SDM Data Definition Language

In this section, we describe a specific *data definition language (DDL)* for the SDM. This DDL allows an SDM schema to be constructed for particular application environments. In the foregoing description of the SDM, we have not relied on a specific DDL syntax, although the discussion has proceeded through many examples which use an example DDL syntax. Although there are many forms of DDL syntax which could be used to describe SDM schemas, we put forth a specific syntax here. The purpose of detailing this specific syntax is to make the specification of the SDM precise. We also believe this syntax is a good one for data base designers to use, and it is used by the interaction formulation advisor prototype (see chapter 7).

The syntax of the SDM DDL is presented in figure 3-7 (which extends over several pages). The notation there is in a Backus-Naur Form style; the particular conventions we use are described at the beginning of the figure. For the most part, the syntax description is self-explanatory; however, the following points are of note:

1. Syntactic categories are capitalized (with no interspersed spaces, but possibly connected by "_"). All lower-case strings are in the language itself, except those enclosed in "*". The latter are descriptions of syntactic categories whose details are obvious (e.g., character strings need not be formally specified).

2. Indentation is an essential part of the SDM DDL syntax. Thus in figure 3-7, the first level of indentation is used for presentation, while all others indicate indentation in the syntax itself. For example, MEMBER_ATTRIBUTE is defined as consisting of an ATTRIBUTE_NAME, followed by ATTRIBUTE_FEATURES (placed vertically below ATTRIBUTE_NAME).

3. Many rules which constrain the set of legal SDM schemas are not accomodated in the syntax shown in the figure. As for any BNF style language description, a variety of context sensitive language features cannot be expressed (as in a programming language where it is not possible to require a variable to be declared prior to its use). For example, in the SDM:

   a. The rule that attributes of different applicability (member attribute, class-determined attributes, and class attributes) must not be mixed is not accomodated in the syntax, as its incorporation therein would be too cumbersome.

   b. A similar statement can be made for the rules that arithmetic expressions must be computed on attributes whose values are numbers, that a common root class must exist for classes defined by multiset operator interclass connections, and so forth.

4. Various defaults are not described in the syntax:

   a. A class is indicated as a nonbase class by the presence of a derivation; otherwise the class is a base class.

   b. The default class kind is "concrete object".

   c. The default is that a class cannot have duplicates; "duplicates-allowed" must be explicitly stated.

   d. An attribute is indicated as derived by the presence of a derivation; otherwise the attribute is primitive, and must have a specified value class.

   e. The default attribute kind is "property".

   f. An attribute is single-valued by default; if it is multi-valued, it must be explicitly declared as such.

   g. An attribute is optional by default; if it is mandatory this must be explicitly stated.

   h. The default is that an attribute is changeable; fixed attributes are explicitly declared as "fixed".

The reader is encouraged to examine the example schema description in figure 3-1, as it was constructed to conform to the DDL syntax rules.

3.5. Operations on an SDM Data Base

An important part of any data model is the set of operations that can be performed on it. The operations defined for the SDM allow a user to derive information from a data base, to update a data base (adding new information to it or correcting information in it), and to include new structural information in it (change the SDM schema).

There is a vocabulary of SDM "base" operations, which are application environment independent and pre-defined. The set of permissible operations is designed to permit only semantically meaningful manipulations of an SDM data base. User-defined operations can be constructed using the primitives. A detailed specification of the SDM operation facilities is provided in chapter 5.

3.6. Summary of the SDM

To summarize the principal features of the SDM, we note that at the coarsest level of detail, an SDM data base is a collection of classes. Each such class is defined in the SDM schema for a given SDM data base. Classes have members, which are entities of one of the following entity types: (concrete) object, (point and duration) event, abstraction, aggregate, and name. A given entity in an SDM data base can be a member of more than one class. Each class has a specified name (unique over all classes in a schema), and an optional description (for documentation purposes). A class is specified as either containing duplicates or not containing duplicates.

SDM classes can be divided into two groups: base classes and derived classes. Base

SDM classes are those that are defined independently, while nonbase classes are those that are defined in terms of other classes. A nonbase SDM class has an associated derivation specification, which consists of an interclass connection. The types of interclass connections provided are: "restrict", to describe a subclass of some class by stating a constraint on its attributes; "subset", for a user-controllable subclass; "extract common members", "extract missing members", and "merge members", for defining a class via a (multi)set operator; "abstract", to define a class whose members are abstractions of the members of some other class; and "aggregate", for specifying a class whose members are collections of the members of some other class.

Each class has a group of attributes associated with it. Like classes, attributes have a name (unique over all attributes of the class), and an optional description. An attribute may describe an aspect of each member of a class (a "member attribute"), an aspect of each member of a class that has the same value for all class members (a "class-determined attribute"), or an aspect of a class taken as a whole (a "class attribute"). A "component" attribute is one which describes some part of a concrete object; a participant attribute logically links an event to participant(s) in that event; a "property" is a general sort of attribute, which is not of one of the preceding two specific types.

A primitive attribute is explicitly given values by a user. Each primitive attribute has a specified value class, which is the class defined in the schema from which values of the attribute must be selected. An attribute is specified as either single-valued or multi-valued. Each primitive attribute is specified as mandatory if it may not have an "unknown" value, and optional otherwise. Similarly, each primitive attribute is defined as fixed if its value cannot change over the lifetime of the entity or class it modifies, and changeable otherwise.

A derived attribute is one whose value is calculated from information in a data base via an attribute derivation specification. The permitted derivation specifications for member attributes allow an attribute to be defined by: "ordering" on other attribute(s); "inversion" of an attribute of some class; "matching" to the members of some class; "existence" of the entity in another class; "merging members recursively" over an attribute; "instances" (the instances of the members of an abstraction class); "constituents" (the constituents of the members of an aggregate class). There are two types of derivation specifications for class attributes: the value of the attribute is the number of (unique) members in the class; the attribute's value is the maximum, minimum, sum, or average of the values of some numeric member attribute taken over the class. To define the value of an attribute in terms of the value of other attributes of the same class and applicability, several inter-attribute derivation specification types are supported which specify an attribute's value as: equal to the value of some other attribute; an arithmetic combination of other attributes; "extract common members", "extract missing members", or "merge members" applied to two or more multi-valued attributes; the maximum, minimum, sum, or average of a multi-valued numeric attribute; the number of (unique) members in a multi-valued attribute; the restriction of the class of values for a multi-valued attribute. In attribute derivation specifications, mappings are used to reference an attribute of the entity which is the value of an attribute, and so forth.

In addition to the attributes listed in the definition of a class, that class may inherit additional attributes from other data base classes to which it is related via interclass connections. An attribute is inherited if and only if the meaning and value of that attribute is the same in the class from which it is being inherited and the class by which it is being inherited.

4. APPLICATION MODELLING WITH THE SDM

In the preceding chapter, a detailed specification of the SDM was provided. In the present chapter, we examine the use of the SDM as a data base application environment modelling tool. We have asserted that the SDM is useful in several ways:

1. It can be used as a formal specification mechanism and data base documentation device. A formal description expressed in terms of the SDM can serve as a communications medium between an application specialist and a data base designer, precisely describing the system for which a data base is to be constructed. It can also be included in the permanent documentation of the data base, to provide a meaningful and unambiguous guide to it.

2. It provides a basis for effective user interface tools, which allow a user to focus on the meaning of the information, and to understand the logical organization of a data base.

3. An application description couched in terms of the SDM is useful to the data base designer as a guide for constructing a conventional data base schema, i.e., one based on a conventional data model.

Specifically, this chapter stresses how the SDM captures the semantic information needed to support each of these applications, and how one models an application environment with the SDM:

1. We examine the ways in which the SDM modelling facilities can be used, and describe how the features of the SDM address the problems of conventional data models (as outlined in chapter 2). The unique aspects of the SDM are stressed.

2. Several important aspects of the SDM as a modelling tool are discussed in detail, viz., the use of derived information and the management of the degrees of the modelling freedom provided in the SDM.

3. We examine the ways in which an SDM data base for a given application environment should be designed. A specific methodology for the design of SDM schemas is presented, and an example data base is included to illustrate its use.

4.1. The Motivation for the SDM Features

Each feature in the SDM is designed to respond to one or more of the observed data modelling problems of conventional data models, as described in chapter 2. In this section, we explain how the motivation for the various SDM features, and analyze how they solve problems with conventional data models:

1. SDM classes are used to model the existence of entities of a given type. Various types of entities are distinguished: (concrete) objects, (point and duration) events, abstractions, aggregates, and names; a distinct kind of class is used to model each of the types of entities that commonly appear in an application environments. The class structure is used to model only the existence of entities. The SDM thus attacks the problems of syntactic data structures and semantic overloading exhibited by conventional data models, in that it allows high level structures to be directly modelled in a schema.

2. Through the mechanism of "event", the SDM allows a data base schema to capture the dynamics as well as the statics of an application environment (in response to the

problem of static orientation). The potentially important semantic differences between objects and events can thus be expressed in a data base's schema.

3. An entity can be a member of more than one class: SDM interclass connections allow nonbase classes to be defined by relating them to others in the schema. For example, a given oil tanker may be a member of SHIPS, OIL_TANKERS, and LIBERIAN_OIL_TANKERS. By supporting the specification of logical interclass relationships, the SDM addresses the problem of disjoint structures exhibited by conventional data models.

4. The notion of a class that contains some of the members of another is important in the SDM. There are several specific ways in which an SDM subclass can be defined:

   a. "Restrict" allows a subclass of a parent class to be defined whose constitution is specified by a predicate on the members of the parent class. This greatly alleviates the problem of nonintegrated subtypes exhibited by conventional data models.

   b. "Subset" allows user-controllable subclasses to be defined, avoiding the conventional problem of dynamic subsets by name.

   c. "Extract common members", "extract missing members", and "merge members" are used to define subclasses in terms of multiset operations on underlying classes.

5. "Second order" entities, i.e., those that are collections (aggregates) or abstractions of others are accomodated in the SDM via the interclass connections "aggregate" and "abstract". In this way information about collections of collections of entities can be captured in a data base schema. The ability to define aggregate and abstraction classes

addresses the conventional data model problems of aggregation by name association and artificial abstraction, respectively. The logical connection between an abstraction and its instances and between an aggregate and its constituents is accomodated via:

a. the "Instances" member attribute of an abstraction class (e.g., the SHIPS that are instances of a given member of SHIP_TYPES),

b. the "Constituents" attribute of an aggregate class (e.g., the SHIPS that are the constituents of some member of CONVOYS),

c. a specification in the definition of an abstraction class of the instances classes that are explicitly defined in the schema as restrictions (e.g., MERCHANT_SHIPS contains the instances of a member of SHIP_TYPES),

d. a specification in the definition of an derived or mixed aggregate class of the constituents classes that are explicitly defined in the schema as subsets]

6. Entity attributes are directly accomodated in the SDM: artificial association entities need not be defined as in conventional data models [Chen 1976, Pirotte 1976]. For example, it is not necessary to introduce an entity to relate ships to their country of registry; rather, this association is captured by the primitive attribute Country_of_registry of SHIPS, and its inversion Ships_registered_here of COUNTRIES.

7. The value class of an attribute can be any class defined in the schema. This avoids the problems of atomic value sets and denotation be selected name. For example, the value class of attribute Captain of class SHIPS is OFFICERS, rather than a class

containing, say, names of officers. By contrast, in the relational and network data models, attribute value classes are required to be atomic "domains", which are atomic sets of data values. It is conventionally assumed that the definition of such domains cannot involve information in a data base, but rather is strictly a data formatting process [McLeod 1977a]. For example, in the relational data model it is possible to state that the value class of a Date attribute is WEEKDAYS, where the members of WEEKDAYS are the strings that stand for working days. However, it is not possible to state that the value of an attribute Tanker is OIL_TANKER_NAMES, where OIL_TANKERS names is required to be those members of SHIP_NAMES that are names of members of OIL_TANKERS. The SDM avoids this problem by allowing the values of attribute Tanker to be members of OIL_TANKER itself.

8. In the SDM, explicit cross referencing across data structures is not necessary as it is in the relational data model; given an entity, a user can directly travel to the entities that are the values of its attributes. For example, the SDM user does not have to cross link ships to their captains by matching on ship name, but rather can directly travel to an officer entity itself via the value of attribute Captain of SHIPS.

9. An SDM data base designer does not need to select a key name around which the data base is to be oriented, i.e., which is to serve as the "primary key" of a relation (file) [Codd 1970]. In response to this problem, the SDM makes it possible to cross reference between entities via any appropriate attribute(s), and SDM classes can have multiple identifiers.

10. SDM entities are clearly distinguished from their names, and the operations of changing an entity and changing one of its names are clearly differentiated. Conventional data models do not adequately accomodate this distinction [Hall 1976].

11. Attributes can be single-valued or multi-valued. This implies that repeating groups can be directly modelled via entity attributes. This eliminates the problem of single-valued attributes exhibited in the relational data model, and obviates the need to introduce artificial entities the purpose of which is to collect together the (potentially) multiple values of an attribute.

12. Attributes with different applicability are distinguished in the SDM: member attributes, class-determined attributes, and class attributes are accomodated. Information concerning the meaning of an attribute, which is missing from conventional models, is thereby captured. For example, the classical quantification problem of differentiating between "all telephones are black" and "telephones that are black" [Wong 1977b] is handled: the former describes a property of the class of all telephones, while the latter refers to the subclass of all telephones that contains those that are black. In the SDM, "all telephones are black" is modelled via a class-determined attribute of class TELEPHONES (with value "black"), while "telephones that are black" is modelled by a restriction of TELEPHONES on the predicate "where Color = 'black'".

13. "Properties", object "components", and event "participants" are specific kinds of attributes that are permitted in the SDM. These correspond to important categories of attributes from the point of view of the user, and the specification of an attribute as one

of these types provides additional semantic information about the meaning of the attribute.

14. The inheritance of attributes in the SDM relieves the conventional data model problem of rigid attributes. The SDM subclass definition facilities and attribute inheritance rules make it possible to associate a member attribute with a class so that the attribute makes sense for each member of that class. For instance, attributes of oil tankers are associated with class OIL_TANKERS, while attributes of all kinds of ships are associated with SHIPS. Since OIL_TANKERS is defined as a subclass of SHIPS, all attributes of ships and their corresponding values are inherited by oil tankers.

15. Derived as well as primitive attributes are supported in the SDM; the value of a primitive attribute is supplied by an explicit data base update, while the value of a derived attribute is calculated from other information in the data base. A flexible vocabulary of derivation specification types is provided in the SDM, to accomodate the most common kinds of derived attributes. Attributes with complex derivations are specified by the definition of one or more related attributes, and by then applying one of the specific derivation specification types to these.

16. Ordering attributes can be directly defined in the SDM, which eliminates the problem of missing ordering capabilities evidenced in conventional data models.

17. "Mandatory" attributes are required to have a value other than "unknown". This makes it possible to require the entities in a class to have some property, i.e., a property the entity must have in order for it to meaningfully be a member of the class.

18. The problem of inflexible degree of binding is addressed in the SDM by allowing a degree of binding to be explicitly specified for each primitive attribute: an attribute can be specificed as "fixed" or "changeable".

19. Because an SDM schema is based on the natural structure of an application environment, the items of information that are likely to be changed by users are in closer correspondence with data base structures than in conventional data models. For example, when a new oil tanker is created, a new member is added to class OIL_TANKERS; all updating is localized to a single entity. By contrast, in the relational data model, the creation of an oil tanker may involve the creation of a new tuple in relation SHIP as well as a new tuple in relation OIL_TANKER.

20. An SDM schema is redundant, in the sense that it contains derived information. This accomodation of redundancy enables the SDM to avoid the problems that appear in conventional data models due to the goal of nonredundancy. Unlike conventional data models, the SDM does not rely on superimposed views for providing different ways of looking at the same data; rather, multiple ways of viewing the same information are intergated into an SDM schema in the form of derived information. The importance of derived information in an SDM schema is discussed in detail in section 4.2.

As described above, an SDM schema captures much more semantic information about the structure of an application environment than the relational, network, or other conventional data models.

4.2. Derived Information and Redundancy

One specific and important way in which the SDM differs from conventional data models and other work on semantic modelling is its accomodation of derived information and redundancy. It has often been argued that relational (and other conventional) data base conceptual schemas should be designed for minimal logical redundancy [Bernstein 1975a, Codd 1970, Codd 1971b, Codd 1971c, Fagin 1977a, Fagin 1977b]. By contrast, the SDM is deliberately a highly redundant data model. Facilities for expressing derived information are an essential part of the SDM; derived information is as prominent in an SDM schema as primitive data.

The essential motivating reasons for integrating derived information into the SDM can be characterized as follows:

1. Different aspects of an application environment are often strongly interrelated, and the same information recurs in different guises.

2. Different users have different perspectives on the same application environment, and a data base schema must be sufficiently flexible to accomodate this variety. A data model should allow the data base design process to balance the requirements of multiple applications into a single data base schema [Raver 1977].

3. Derived information that will be frequently extracted from a data base should be as easy and natural to access as the primitive information represented in that data base. Allowing derived information in a data base schema also permits the schema to adapt to changing uses: as data base usage patterns evolve, new derived information can be

added to the schema.

4. Conventional data models rely on the use of multiple "external schemas" [ANSI/X3/SPARC 1975] or "views" [Chamberlin 1975, Chang 1975] to accomodate derived information. Here, a "conceptual schema" is defined for a data base, and multiple "external schemas" are constructed as subsets [ANSI/X3/SPARC 1975] or simple reorganizations of it [Chamberlin 1975, Stonebraker 1975b]. There are a number of problems with this approach:

a. In effect, an external schema (view) serves as a retrieval transaction on a data base; it cannot introduce both primitive and derived information. For example, it is not possible to define a view consisting of all oil tankers so that it is a subrelation of the relation containing all ships and so that its members have additional attributes (that other ships do not); thus information on oil tankers must be placed in separate logical locations in a data base, which may make it hard for users to find.

b. Since views are superimposed on top of a conventional data base, information concerning their definition is not present in the data base schema, but is rather external to it. This means that such information is not integrated into a common form (the schema), where its consistency and completeness can be assessed.

c. It may be hard for a user to find an appropriate view. Instead of examining an integrated, structured data base, he must step though the view definitions to try to find a useful item of derived information.

4.3. Modelling Degrees of Freedom

As we have seen, the SDM provides a variety of constructs for use in constructing a schema to model a given application environment. This variety is critical to the goal of capturing the natural structure of an application system in an SDM schema. .However, the degrees of modelling freedom provided in the SDM are limited, in that we have attempted to provide the necessary modelling flexibility without introducing a great number of features that serve the same functions. In sum, the SDM suggests alternative ways of modelling some given data, but does not force the user to employ some single and possibly unnatural modelling mechanism.

4.3.1. Modelling Flexibility

As indicated above, there are a number of degrees of freedom available in constructing an SDM schema. In sections 4.5 and 4.6, we examine how these degrees of freedom are used in the data base design process. In this section, we present an example SDM modelling flexibility: the modelling of relationships among entities.

In the SDM, an association among entities can be modelled in the following ways:

1. Attributes can be used to establish relationships between the entities. A bidirectional association can be expressed as an attribute and an inversion derived from it.
2. A (duration) event can be used to model an association that has an abstract existence as an entity. This may be the case if the association entity is to have attributes of its own, or if it is to be used as the value of another attribute. For example, if the concept

of an "assignment" (of a captain to a ship) is of interest in the application environment, the duration event class ASSIGNMENT can be defined (as in the TMDB).

3. Higher order (e.g., ternary) associations can be modelled by (duration) events, by means of several binary associations, or by using multi-valued attributes.

Although the choice of which of these methods is to be used for a given association is left to the data base designer, the definition of the SDM constructs themselves suggests the appropriate modelling method. The SDM data base designer is not directly confronted with the problem of how to model an "association". Rather, the SDM encourages the designer to think in terms of events and attributes. Attributes are used to relate entities to others, and duration events are used to describe "association entities", i.e., entities that serve to relate others. It is important that the designer think *in terms of the* "event" - a natural application environment concept, rather than the artificial concept of an "association entity".

Attributes that are semantically related to one another in an SDM schema are defined by declaring one of the attributes to be primitive and the other derived. For example, the attribute Tanker of INSPECTIONS is primitive, while its inversion, Inspections of OIL_TANKERS is derived. As stated above, this implies that attribute Tanker of INSPECTIONS is explicitly given values by users, while the value of Inspections of SHIPS is calculated from other information in the data base. Distinguishing primitive and derived attributes is important: it allows the specification of whether or not it makes sense to update an attribute. If it is not possible to determine which member of an attribute pair is primitive and which is derived, then an (duration) event should be used to model the association.

4.3.2. The Control of Derived Information

Since the SDM provides a substantial number of options for modelling, it is important that the designer be provided with some guidance in selecting the correct construct to model a given piece of information. Moreover, there must be an effective ways of handling the complexity that is a consequence of the richness of the SDM, e.g., as it impacts the understandability of a data base. The SDM manages these problems of complexity and guidance in the following ways:

1. The SDM limits the number of choices of modelling each aspect of an application environment, e.g., the SDM provides a small number of ways of defining classes and attributes. A rich set of mechanisms for defining derived information in a schema is also provided, but the number of such mechanisms is limited for manageability.
2. Design heuristics can be provided for the data base designer, in order to guide him in making modelling choices and in designing an SDM schema.

In the SDM, derived classes are defined via one of a few types of interclass connections; the SDM provides a set of interclass connections that are designed to accomodate most of the classes needed to model an application environment. Specifically, we stop short of a full "case grammar" analysis [Schank 1973] or similar approach to the general knowledge representation problem. This is a consequence of our desire to provide a model that includes important semantic distinctions, but which is not unmanageably complex.

Arbitrary transactions (data selection algorithms) are not permitted to materialize derived classes. This allows the basic structuring of classes to remain relatively simple; since classes are the principal components of an SDM data base schema, we wish to restrict the facilities used to define them to a small number of simple structuring capabilities. Complex redundancy relationships are to be expressed in attribute definitions, rather than in class specifications.

The SDM provides a wide variety of attribute derivation specification types. This vocabulary of derivation specification types is designed to directly accomodate the majority of the kinds of useful attributes in most application environments. For attributes with complex derivations, the primitive derivation types can be used in a stepwise fashion to build intermediate attributes, which are later used in the derivation specification of the desired attribute.

Derived classes with arbitrarily complex derivations can be defined by using one of the interclass connections along with derived attribute(s). For example, to define a restriction of a given class with a complicated predicate, we may define any derived attribute(s) that are needed and then use the interclass connection "restrict" to define the desired class. Consider the class TEN_MOST_RECENT_INSPECTIONS. This class can be defined by including an ordering attribute for inspections (e.g., attribute Order_by_date defined on decreasing value of Date of INSPECTIONS), and then defining a restriction with the predicate "where Order_by_date <= 10".

4.4. Relationships to Other Data Models

In chapter 2, we indicated how the SDM differs from other higher level data models, particularly those that have been designed with goals related to the goals of the SDM [Chang 1975, Chen 1976, Chen 1977a, Pirotte 1976, Pirotte 1977, Schmid 1975a, Wiederhold 1977]. Now that the SDM has been described in detail, it is possible and useful to look more closely at the differences between the SDM and the work that we see as most closely related to ours. In particular, we shall focus on the recent work of Smith and Smith [Smith 1977a, Smith 1977b, Smith 1977c, Smith 1978a].

Smith and Smith have presented a design methodology for relational data bases based on application semantics. As we indicated in chapter 2, several ideas from their work have given rise to features in the SDM. However, there are a number of ways in which our work on the SDM differs from theirs:

1. The SDM is not closely tied to relational data bases. The SDM is not merely a tool for designing relational data base, but rather is a complete high level data model itself.

2. The SDM does not strive for minimality of modelling constructs. Rather, it provides a rich collection of data structure types. SDM data structures are much more highly interrelated that the Smiths'. For example, we provide a number of different types of interclass connections each of which relates two or more classes in a different way.

3. One of the apparent points of similarity between the SDM and the Smiths' work is the concept of "generalization". (Referred to as "abstraction" in the SDM.) The basic

idea of "abstraction/generalization" is similar in these two approaches, but there are some significant differences, including:

a. in the SDM, the instances classes of an abstraction class do not necessarily form a partition [Lee 1978a], while the Smiths' concept of generalization is based on partitions,

b. the SDM allows the instances classes corresponding to an abstraction to be explicitly identified in the schema.

4. The SDM takes the approach of allowing entities to stand for themselves, rather than requiring some logical key (identifier) to be selected to stand for them (e.g., as the value of an attribute).

5. The Smiths indicate that it is inappropriate to assign a fixed interpretation to a conceptual schema: "A representation for conceptual models which requires a fixed interpretation will artificially constrain concept integration. Artificial constraints usually manifest themselves in increased complexity and anomalous update properties." [Smith 1978a]. We agree that it is essential to support different ways of looking at the same information; in fact, this is one of the reasons that derived information is accomodated in the SDM. However, it is also essential at some point to integrate various user views into a common form; this integration is accomodated in the SDM via the definition of nonbase classes and derived attributes. Varied user views can still be supported on top of an SDM data base to allow for varied ways of looking at the same data.

6. As described above, the SDM has been designed with specific applications in mind.

Thus, for example, the design of the SDM interaction formulation advisor (IFA) has had a significant impact on the SDM itself (see chapter 7). This influence has arisen from our current experience in building and experimenting with a prototype IFA.

4.5. The Design of SDM Data Bases

The first four main sections of this chapter have stressed the available types of structural primitives in the SDM, and how the can serve as effective modelling facilities. However, we have not directly addressed the problem of how these primitives are used to construct a data base for a given application environment. This section and the next focus on the issues of designing SDM data bases.

The problem of *SDM data base design* has two levels:

1. The process of *SDM schema design* consists of the construction of a data definition language description of the schema, such as the TMDB schema. This involves the definition of appropriate classes, attributes, etc. for the application.

2. If the SDM is being used to aid in the design of a conventional data base, the SDM constructs must then be translated into those of the conventional data model. A standard and perhaps automated means of accomplishing this translation could be used. Alternatively, the data base designer could manually perform this mapping, possibly with the aid of design guidelines and heuristics. It is important to note that an SDM schema captures much more semantic information about a data base application environment that does a conventional schema; this means that information will be lost

when mapping SDM constructs into those of a conventional data model. The issues of *SDM schema translation* also arise in a DBMS that supports the SDM and that relies on a conventional DBMS to manage physical storage and access.

In the remainder of this chapter, we address the problem of SDM schema design, and defer the problem of schema translation. We are presently working on the latter problem, but its detailed consideration is beyond the scope of this thesis.

4.6. SDM Schema Design

A considerable amount of work has been done on the problem of "logical" data base design, i.e., on designing conceptual schemas, and on developing techniques useful in the logical data base design process [Adiba 1976, Benci 1976, Beriid 1977, Bernstein 1976, Bracchi 1976, Brown 1975, Bubenko 1977b, Chen 1977a, Codd 1971b, Codd 1971c, Cook 1975, Fagin 1977a, Falkenberg 1976b, Lindgreen 1974, Lyon 1971, Moulin 1976, Nijssen 1974, Schmid 1975, Senko 1975d, Smith 1978a, Weiner 1977, Wiederhold 1977, Yeh 1977]. Although we do not present a full and complete treatment of the problem of SDM schema design in this thesis, it is important to highlight the fact that the SDM provides a basis for a logical data base design methodology that we believe has advantages over most of those recently proposed.

The purpose of this section is therefore to demonstrate how a schema for an SDM data base can be constructed. A design methodology is presented, which imposes some structure on the schema design process. It serves as a guide to the data base designer; the designer uses this guide together with his knowledge of the semantics of the application environment and the nature of the uses to which the data base will be put to produce an SDM schema. The design methodology also provides a framework for presenting the most important design principles for the SDM (*SDM design heuristics*). These design heuristics, along with the specifics of the SDM itself, provide the principal basis for the design procedure.

In order to make the discussion of SDM schema design concrete, a new example application environment is presented, and an SDM schema designed for it. This application domain concerns aircraft maintenance (the "aircraft maintenance application environment" - AMAE), and it specifically deals with maintenance job scheduling, parts inventory, employee job assignment, and so forth. In the following section, it is assumed that a hypothetical data base designer is constructing an SDM schema for the AMAE. The SDM schema resulting from the design process followed by this designer is shown in figure 4-1, and is described in the next section.

4.6.1. An SDM Schema Design Methodology

This section provides a description of the SDM schema design methodology. There are a number of steps involved in the schema design process, and the designer follows these in order. However, the SDM schema design process is not entirely sequential, and there are a number of places in the methodology at which the designer may need to modify his actions in a previous step; these places are pointed out below. Nonetheless, the methodology above does provide considerable guidance for the data base designer. We also note that the design methodology presented here is oriented to the design of an initial SDM schema, which will subsequently evolve.

The steps of the SDM schema design process are as follows:

1. First, the designer prepares an initial list of the collections of entities (classes) that are to be present in the SDM data base (*class list*). The specific nature of this list depends on the designer's understanding of the application environment, but he should include in the class list the things of interest (e.g., objects and events) that appear in the application environment. Such entities can often be found by determining what things have attributes of interest. Considering anticipated types and units of data base updates is another useful way to determine which classes are to be included in the class list; for example, if a common database update is the recording of information on the completion of maintenance jobs on aircraft, it is probably the case that a class of "jobs" should be included in the schema.

   The designer prepares the class list by providing a carefully selected name for each class of interest. He also optionally provides a textual description of each class, for documentation purposes.

   For the AMAE, the designer identifies the classes MECHANICS (all employees who are concerned directly or indirectly with aircraft maintenance), AIRPORTS, B727 (all 727s), B707, D10, L1011, JOBS (maintenance actions on aircraft), SCHEDULED_JOBS, BREAKDOWNS (aircraft breakdowns), PART_TYPES (types of parts for aircraft repair), PART_STOCKAGES_BY_AIRPORT (the parts on hand at an airport), PART_OUTTAGES (occurrences of airports running out of parts of a given type), and PROBLEMS.

2 The class list is now examined to determine the subclasses of interest. The designer determines which of the classes in the class list are subclasses of other classes in the list, i.e., which classes contain some of the members of others; for each such subclass, the designer records the name of the most immediate parent class. Since he is thinking about subclasses, the designer should add to the class list any subclasses of classes in the list that are of interest. To do this, he considers in turn each class in the list, and determines if there are any kinds of entities that are specific subtypes of those modelled in the class he is considering; appropriate subclasses are added to the class list, and the name of their most immediate parent class is recorded. Any new classes are described as in step 1.

For the AMAE, SCHEDULED_JOBS is identified as a subclass of JOBS, and PART_OUTTAGES is identified as a subclass of PART_STOCKAGES_BY_AIRPORT. The following are added to the class list: subclasses JOBS_PERFORMED and BREAKDOWN_JOBS (jobs in response to aircraft breakdowns) of JOBS, and the subclass PART_TYPES_OUT_OF_STOCK_SOMEWHERE (parts that are out of stock at some airport) of PART_TYPES. Also, the designer considers the possibility that a superclass of two or more classes is of interest. In the AMAE, the superclass AIRCRAFT of classes B727, B707, D10, and L1011 is of interest. A subclass of

AIRCRAFT, namely AIRCRAFT_DUE_FOR_SERVICE, is also added to the class list.

3. The class list is scanned to determine which, if any, of its elements are abstractions or aggregations of other classes in the list. Such classes are often of interest if their members have attributes of their own. If any classes of this kind are present in the class list, the class underlying the abstraction or aggregation is recorded. Any further classes of interest that contain abstractions or aggregates of the members of some class in the class list are added to that list, and specified as in step 1.

For the AMAE, the class AIRCRAFT_TYPES (kinds of aircraft, e.g., B707, D10, etc.) is added as an abstraction of AIRCRAFT, since it is of interest, and since the designer wants to record the engine type and number of engines for each aircraft type. The abstraction class of MECHANICS named MECHANIC_QUALIFICATION_GROUPS (groups of employees by qualification) is also of interest.

4. The designer next produces a list of the aspects of interest for each class in the class list. He considers aspects of each member of a class, as well as aspects of a class as a whole. This is a preliminary specification of the member attributes, class-determined attributes, and class attributes of interest for each class. Specifically, the designer provides a name for each attribute, an optional description of its meaning, and specifies whether the attribute is single-valued or multi-valued. For example, class BREAKDOWNS has member attributes Aircraft, Date, Problem, and Job (the job fixing the breakdown). The designer also specifies the applicability of the attribute: a

member attribute is one that has a value for each member of the class; a class-determined attribute describes an aspect of each member of the class and has the same value for each member; a class attribute describes a property of a class taken as a whole.

Care should be exercised to associate an attribute with the most appropriate class. For example, since only jobs that have been performed have a "date performed", the member attribute Date_performed is associated with class JOBS_PERFORMED rather than with class JOBS. If necessary, a new class may be introduced into the class list to accomodate an attribute whose logical place is with that new class, e.g., if JOBS_PERFORMED were not in the class list, it would be added thereto in order to accomodate attribute Date_performed. Each such new class is specified as in step 1, and steps 2 through 4 are performed for this new class.

5. For each attribute identified in step 4, the class from which its values are to be selected is identified. The designer examines the class list to make a selection; if an appropriate class is not in the list, an addition to the list is necessary (and steps 1 *through 4 are repeated for it). In identifying the value class of an attribute,* the designer should try to be as specific as possible, so long as the value class is itself meaningful and of interest. For example, the value class of attribute Scheduled_jobs of AIRCRAFT is specified as SCHEDULED_JOBS rather than JOBS, because only scheduled jobs are permitted as a value of this attribute. Moreover, classes other than name classes should be used except where names are really best. For example, aircraft

names should not be used where aircraft are intended. For the AMAE, the designer selects AIRCRAFT as the value class of attribute Aircraft of class BREAKDOWNS, and BREAKDOWN_JOBS as the value class of attribute Job.

6. Derivation specifications for all subclasses in the class list are now formulated. For each class C in the class list:

a. If members of the parent class are to be added to and deleted from the subclass via direct user control, then the "subset" interclass connection is used. For example, the derivation of the class AIRCRAFT_DUE_FOR_SERVICE is "subset of AIRCRAFT".

b. It may be the case that the subclass C can be defined as containing:

i. the members that two or more other classes (Cl, C2, ...) have in common,

ii. those members that are in either of two (or more) other classes (Cl, C2, ...),

iii. the members of some other class Cl that are not in some class C2.

If so, then "extract common members", "merge members", or "extract missing members" (respectively) is used to define C.

c. If the members of the subclass C can be defined by means of a predicate on the member attributes of the parent class or specified to be those members of the parent class that are the value of some attribute of some other class, then the interclass connection "restrict" is used. For example, the class JOBS_PERFORMED is defined via "restrict JOBS where Status = 'complete'". New member attribute(s) of the parent class may be required to allow the restriction predicate to be formulated; any new attributes are specified by repeating steps 4 and 5 for them. For example, in the TMAE, the designer adds attribute Status to the list of member attributes of JOBS in order to allow the class JOBS_PERFORMED to be defined.

In defining subclasses, it is necessary to avoid circular definitions, e.g., defining C1 as a subclass of C2 which is in turn defined as a subclass of C1. Furthermore, a subclass should be defined via a multiset operator interclass connection if it is possible to do so using other classes that are of interest. This is desirable because it allows the subclass to inherit attributes from all of the classes involved. Figure 4-2 contains an example taken from the TMAE that shows that class MILITARY_OIL_TANKERS can be defined as a restriction of OIL_TANKERS, as a restriction of MILITARY_SHIPS, or as a restriction of SHIPS. The design heuristic invoked here is that if the classes OIL_TANKERS and MILITARY_SHIPS are of interest, then class MILITARY_OIL_TANKERS should be defined in terms of them. Moreover, multiple levels of subclasses should only be defined when the class at each level is of interest. For example, if only one of OIL_TANKERS and MILITARY_SHIPS is of interest, then MILITARY_OIL_TANKERS should be defined as a subclass of it; if neither of these is of interest, then MILITARY_OIL_TANKERS is best viewed as a restriction of SHIPS.

7. Class derivation specifications for each abstraction and aggregation class are stated. For an abstraction class, a grouping on the members of the underlying class is specified. For example, the class AIRCRAFT_TYPES has the derivation "abstract AIRCRAFT

on common value of Type", and the class MECHANIC_QUALIFICATION_GROUPS is defined by "abstract MECHANICS on common value of Aircraft_types_qualified_on". The former abstraction class is a "type abstraction", in that it contains members which are "types" of the members of another class; here, as in all "type abstractions", the instances classes of AIRCRAFT_TYPES partition the underlying class. The latter abstraction class, MECHANIC_QUALIFICATION_GROUPS, involves a grouping on the value of a multi-valued attribute, which means that the instances classes of MECHANIC_QUALIFICATION_GROUPS do not partition the underlying class.

Constraints are placed on the permissible form of grouping expressions in the definition of abstraction classes, in order to avoid circular definitions. In particular, it is not permissible for the grouping expression used to define an abstraction class C to involve an attribute A that is itself defined in terms of C. This implies that A cannot have values in class C. For example, it does not make sense to select AIRCRAFT_TYPES as the value class of attribute Type of AIRCRAFT. When an abstraction class is being defined, it may be necessary to introduce a new attribute to allow an appropriate grouping expression to be stated; for example, attribute Type of AIRCRAFT was added by the designer to allow AIRCRAFT_TYPES to be defined by "on common value of Type" (the value class of Type is AIRCRAFT_TYPE_NAMES). Any new attributes are specified as in steps 4 and 5 above.

After an abstraction class is defined, the designer checks to see if any of its instances classes are explicitly defined in the data base as restrictions of the class underlying the abstraction. If so, they are listed in the "defined instances classes" clause of the abstraction derivation specification. For the AMAE, the designer might have observed that B727, B707, D10, and L1011 are the explicitly defined instances classes for the abstraction class AIRCRAFT_TYPES. In fact, the designer decided that B727, B707, D10, and L1011 are only interesting as members of the abstraction class, and are not directly of interest as classes in the schema. So, they are removed from the list of data base classes.

The derivation specification of each aggregate class is specified in an analogous way to an abstraction class derivation. For aggregates, the important choice of which type of aggregate is appropriate must be made (primitive, derived, or mixed). A primitive aggregate is used if none of the constituents classes of the aggregate are to be explicitly defined in the schema; a derived aggregate is selected if all of the constituents classes are to be defined as subsets of the class underlying the aggregate; a mixed aggregate class is used if some but not all of the constituents classes are defined in the schema.

8. For every class, each of its attributes is determined to be either primitive or derived. Attributes whose values are to be directly modified (updated) by users are defined as primitive, while those that are not to be updated are specified as derived. For each primitive attribute, its value class is simply specified as the class which was identified in step 5 as the class from which the values of the attribute are to be selected.

Each primitive attribute can also be optionally specified as either mandatory or optional. An attribute is mandatory if it cannot have an "unknown" value. (The default is optional.) Each primitive attribute can also be optionally specified as either fixed or changeable. (The default is changeable.) A fixed attribute is one whose value cannot change during the lifetime of the entity that it modifies. For example, attribute Id of MECHANICS is "fixed", since by definition, a mechanic's identifier cannot change during his employment.

Each derived attribute is assigned a derivation specification. The designer selects an appropriate derivation type by examining the list of possible derivation specifications: for member attributes he choses from the list of member attribute derivation types, for class-determined attributes he choses from the list of class-determined derivations, and similarly for class attributes. For example, the designer determines that the member attribute Job of BREAKDOWNS is to be defined by inversion, i.e., it has the derivation "invert Breakdown of BREAKDOWN_JOBS". During the process of selecting an appropriate derivation type, the designer is guided by the knowledge of which class contains values of the attribute (from step 5): the designer stated in step 5 that the values of attribute Job of BREAKDOWNS are in the class BREAKDOWN_JOBS, which limits the types of attribute derivations that he must consider (e.g., he rules out arithmetic expressions on other attributes).

If no appropriate derivation type can be found, auxiliary attribute(s) may need to be defined. Any new attributes that are needed are defined as in steps 4 and 5.

9. Any classes that have not been defined as nonbase are now defined as base classes.

The includes specifying the "kind" of each base class. If a class consists of names, i.e., if its details are not of interest and if its members are most naturally modelled as names (e.g., Id of MECHANICS), then a class is marked as a name class and considered in step 10. All other base classes are specified as containing one of the following kinds of entities: "object", "event", "concrete object", "point event", "duration event". When the more specific terms are appropriate ("concrete object", "point event", "duration event"), they should be used, but when the specificity is not needed, the more general term should be used. For example, class MECHANICS is considered a concrete object class, while JOBS is an event class.

The unique identifiers are also determined for each of these base classes. The designer specifies groups of one or more member attributes, so that the attribute(s) in each such group uniquely identify the members of the class. For example, class JOBS has the identifier Job_number. In selecting unique identifiers, the designer should avoid using large groups of attributes, e.g., identifying mechanics by the combination of their name, zip code, and age (assuming this information is in the schema). Rather, identifiers that include one or a few attributes should be used. Attributes that are termed "names", "id numbers", and the like are good candidates.

The designer also determines for each base class, whether or not it is to contain duplicates. (The default and normal case is that a base class does not contain duplicates.)

10. Name classes identified in step 9 are now considered in detail. Each name class is

defined as a subclass of STRINGS or NUMBERS. In the AMAE, the following name classes are defined: ID_NUMBERS, PERSON_NAMES, AIRPORT_NAMES, AIRCRAFT_NUMBERS, ENGINE_TYPE_NAMES, AIRCRAFT_STATUS_LEVELS, DATES, TASK_NAMES, JOB_NUMBERS, JOB_STATUS_NAMES, PART_NUMBERS, TEXT, and PROBLEMS.

The derivation specification of a name class is determined first by noting whether the class in question is a subclass of STRINGS or of NUMBERS. The derivation specification is then one of the following:

a. "Restrict" is used in the derivation specification for a name class that consists of those members of STRINGS or NUMBERS and which possess some particular property. For example, one might define DATES as all strings of the form "8/21/78". The restriction predicate can specify the pattern of the class members, as discussed in chapter 3, or it can be null, which means that all strings or numbers are members of the class being defined. For simplicity, the name classes shown in figure 4-1 have null restriction predicates.

b. "Subset" is used to define a subclass of STRINGS or NUMBERS whose members are to be enumerated. For example, class TASK_NAMES is defined as containing a specified list of members (e.g., "repair engine", "service hydraulic system", etc.).

c. "Extract common members", "extract missing members", and "merge members" are used to define a name class in terms of other name classes.

This completes the design process, and an SDM schema has now been specified. The syntax of the SDM data definition language presented in figure 3-7 is used in the schema. Figure 4-1 contains the schema for the AMAE; this schema was produced using the design methodology presented above.

The design process for the TMAE took about two hours for the author, including the time spent in debugging it. This debugging was accomplished with the aid of the SDM schema analyzer, a component of the prototype interaction formulation advisor (see chapter 7). The analyzer examines a schema to check it for legality, e.g., to make sure that all referenced classes and attributes are defined, to check for circularities in definitions, and so forth. Of course, the author is hardly a random designer, and an intimate knowledge of the SDM made the design process easier.

We are presently investigating the problem of SDM schema design in more detail, and plan to further discuss SDM data base design aids in forthcoming documents. We specifically plan to accentuate the methodology and improve it so that random designers can in fact use it to easily design SDM schemas.

4.7. Modelling Limitations

The SDM has been designed to capture the most important semantic constraints associated with common data base application environments. However, to capture all of the possibly useful types of semantic restrictions in a schema, the SDM would be augmented with an additional mechanism such as those proposed in the context of the relational data model [Eswaran 1975, Eswaran 1976a, Hammer 1975a, Hammer 1976d, McLeod 1976e, Stonebraker 1974b]. This kind of supplementary SDM "semantic integrity constraint" mechanism would allow arbitrary predicate calculus-like statements to be asserted on a data base. These assertions would be required to be true on the state of a data base or on transitions between data base states. When a user attempts to modify the data base, the assertions that may be violated if the specified data base update were performed are checked with respect to the update. If none of these assertions is violated, then the update is performed, otherwise a *violation action* is executed. This violation action may be to reject the specified update, or to take some other action (such as notifying some appropriate user of the situation).

To a first approximation, we believe that the violation actions normally associated with the semantic constraints built into the SDM involve preventing their violation by rejecting any data base update that would cause them to not be satisfied. Many of the additional sorts of constraints one may wish to impose via a supplemental semantic integrity constraint mechanism would have other kinds of violation actions.

In the context of the SDM, a supplemental semantic integrity constraint mechanism would be used to capture relationships among the data that do not lend themselves to a more structured canonical description. We stress however that because of the of the richness of the SDM, most important semantic information can be captured directly in the schema; the use of such a semantic integrity constraint mechanism would thus be limited.

We provide here several examples of semantic constraints that a supplemental semantic integrity mechanism would accomodate (examples are selected from the tanker monitoring data base (TMDB)):

1. The designer may wish to assert a constraint on the number of members in a class, the size of a multi-valued attributes, and so forth. For example, one might require each oil tanker to have at least two engines. This constraint can be imposed in the SDM, but not in the most natural way: one can define two member attributes of OIL_TANKERS, each with value class ENGINES, and declare both as mandatory. The problem with this way of expressing the constraint is that it requires two single-valued attributes to be used when a multi-valued attribute is actually desired.

2. The values of a member attribute may be required to:

   a. exhaust the value set of the attribute,

   b. have a size within some specified bounds (e.g., a multi-valued attribute required to have a given number of values),

   c. be non-overlapping for the member of the class (viz., each value can be used at most once).

3. Additional constraints can be placed on inherited attributes, e.g., that the value of a member attribute inherited from one class to another must be in present in a subclass of its value class. For example, in the TMDB, attribute Involved_ship of OIL_SPILLS is inherited from INCIDENTS; it may be desirable to require the value of Involved_ship of OIL_SPILLS to be present in OIL_TANKERS (a subclass of SHIPS, the value class of Involved_ship of INCIDENTS).

These kinds of semantic constraints might be useful in modelling certain types of information in application environments, but they are not currently directly captured in the SDM. We are presently studying the implications of embedding some of these constraints in the SDM itself. However, it is important to note that we will continue to exercise great care in controlling the features of the SDM so as to retain a manageable data model.

We close this section by noting that there are two main advantages of the SDM over the relational data model vis-a-vis semantic integrity constraints:

1. The most important kinds of constraints are captured directly in the SDM data structures and the interconnections among them. Specifically, interclass connections express relationships between classes that must be captured in the relational model by means of constraints (e.g., that all oil tankers are ships).

2. The SDM provides natural places to attach supplemental constraints. In the relational model, an additional mechanism is thus needed to specify the subset of tuples in a relation to which a constraint is to apply (the "constrained collection" of Hammer and McLeod [Hammer 1975a, Hammer 1976d]); but in the SDM, nonbase classes provide a direct place to attach constraints. For example, in the SDM, a constraint on all oil tankers would be attached to class OIL_TANKERS, while in the relational data model, such a constraint would probably be attached to a relation containing tuples for all ships (because of its lack of "subtype" definition capabilities).

5. OPERATIONS ON AN SDM DATA BASE

The data in a data base structured in terms of the SDM is the *information content* of that data base. In order for users to be able to examine and modify the information content of an SDM database, they must be able to invoke operations on that data base. In this chapter, we specify the *SDM transactions*, which are the permissible operations that are invoked by users to manipulate an SDM data base.

Specifically, a transaction is an operation that manipulates an SDM data base in one or more of the following ways:

1. selects data from a data base (retrieval),
2. adds information to a data base,
3. corrects information in a data base,
4. alters the schema of a data base (adding new structural information).

5.1. Base and User-Defined Transactions

A set of operations is built into the SDM: the SDM *base transactions*. Base transactions can be applied to any SDM data base to manipulate it, i.e., they are operations that can be meaningfully applied to any SDM data base, and are independent of the application environment.

Base transactions are also used as constituents of *user-defined transactions*. User-defined transaction are specified in the SDM *interaction formalism*, which is the data manipulation language of the SDM. Such transactions are used to capture manipulations of a data base that are specific to the application environment modelled in the data base.

In the remainder of this chapter, we present the SDM base operations. This is followed by a description of the SDM interaction formalism. Finally, we discuss how the SDM transaction facilities can be integrated with the capabilities of an "external" programming language.

5.1.1. The SDM Base Transactions

The SDM base transactions correspond to semantically meaningful manipulations of an application environment. Specifically, there are four main types of SDM base transactions, according to the way they are used:

1. *Data selection transactions* are used to extract some information from a data base. Data selection operations are used to extract information from a data base; they assume that a data base is an accurate model of the application environment and embody a query of the data base.

2. *Recording transactions* are used to record new information in a data base; they record changes in the application environment.

3. *Structural modification transactions* are used to alter the structure of a data base. In essence, structural modification transactions effect changes to the schema of an SDM data base. Data selection transactions are often used in conjunction with structural modification transactions to describe derived information to be added to the schema. As we have said, the act of retrieving information from a data base and the process of

defining information in a data base are essentially the same. Structural modification transactions thus define new data base structures, and add new derived information to a schema.

4. *Output transactions* are used to transmit data in an SDM data base to an external medium. Data selection transactions specify new derived information of interest, and output transactions actually transmit this information to users. For example, output transactions support the transfer information from SDM classes to a conventional file system, and allow it to be printed it on a console or other output device.

5.2. Base Data Selection Transactions

Base data selection transactions allow data to be extracted from an SDM data base; that is, they allow derived information to be defined. Specifically, there are two main types of base data selection transactions:

1. An *interclass connection transaction* defines a new class using an SDM interclass connection.

2. A *derived attribute transaction* defines a new derived attribute that is of interest.

These two types of base data selection transactions are specified in detail in sections 5.2.1. and 5.2.2.

5.2.1. Interclass Connection Transactions

The SDM interclass connections as specified in chapter 3 are transactions, since they define derived classes; that is, they can extract information from a data base. An interclass connection that allows a new class to be defined strictly in terms of the data base schema is called a *closed transaction*. The interclass connections "restrict", "abstract", "extract common members", "extract missing members", "merge members", and "derived aggregate" are closed, since they require that a user provide only a definition of the new class via a predicate (for a restriction), a grouping expression (for an abstraction), a list of classes (for a class defined by a multiset operator or a derived aggregate class).

The interclass connections "subset", "primitive aggregate", and "mixed aggregate" are not closed: in order to define a class using these interclass connections, a user must examine the members of classes. For example, if a user specifies that he wishes to define a subset of a class in the data base (using the "subset" interclass connection), he will need to specify which of the members of the parent class are to belong to the subset. Since the subset is not determined by information in the data base (or a restriction would be used to define it), the user will need to step through the members of the parent class and identify which of these belongs to the subset. Similarly, for a derived or mixed aggregate, a user must identify the constituents of each member of the aggregate class.

An interclass connection transaction is invoked as follows:

```
CLASS_NAME <-
   INTERCLASS_CONNECTION_INVOCATION
```

Here, CLASS_NAME and INTERCLASS_CONNECTION_INVOCATION are syntactic categories. In using the above template to define a specific interclass connection transaction, CLASS_NAME is replaced by the name of a new class, viz., the class being defined by the transaction. INTERCLASS_CONNECTION_INVOCATION is replaced by an instantiation of an interclass connection, written exactly as it would appear in a class derivation specification. For example, the class containing all double-hulled oil tankers would be defined as follows:

```
DOUBLE_HULLED_OIL_TANKERS <-
    restrict OIL_TANKERS
        where Hull_type = 'double'
```

5.2.2. Derived Attribute Transactions

A derived attribute transaction defines an augmented version of an SDM class, by defining a new derived attribute of that class. The attribute derivation specification types (as specified in chapter 3) are used in derived attribute transactions, as follows:

```
CLASS_NAME (augmented) <-
    ATTRIBUTE_DERIVATION_INVOCATION
```

Here, CLASS_NAME is a syntactic category; to construct a specific transaction, it is replaced by the name of the class for which a new attribute is being defined. ATTRIBUTE_DERIVATION_INVOCATION is of one of the following forms:

```
define derived member attribute ATTRIBUTE_NAME
    as MEMBER_ATTRIBUTE_DERIVATION

define derived class-determined attribute ATTRIBUTE_NAME
    as CLASS-DETERMINED_ATTRIBUTE_DERIVATION

define derived class attribute ATTRIBUTE_NAME
```

as CLASS_ATTRIBUTE_DERIVATION

In the above, MEMBER_ATTRIBUTE_DERIVATION, CLASS-DETERMINED_ATTRIBUTE_DERIVATION, and CLASS_ATTRIBUTE_DERIVATION are specified exactly the same as the corresponding attribute derivation specifications would be expressed in an SDM schema. For example, the attribute Commission_ordering can be defined as follows:

```
OFFICERS (augmented) <-
    define derived member attribute Commission_ordering
        as order by decreasing Date_commissioned within Country
```

Commission_ordering describes the sequential position of each member of OFFICERS by Date_commissioned, within each country.

5.3. Base Recording Transactions

Base recording transactions are used to record new information in an SDM data base. In particular, base recording transactions are used to change the contents of the classes of a data base, e.g., to add new members to classes or to change the values of attributes of members of classes.

There are three main types of base recording transactions:

1. *class recording transactions*, which modify the contents of a class,

2. *attribute recording transactions*, which alter the values of attributes of the members of a class or the class as a whole,

3. *error correction transactions*, which correct errors made in the recording of information

in the data base.

Sections 5.3.1. through 5.3.3 detail the SDM base recording transactions.

5.3.1. Class Recording Transactions

A class recording transaction modifies the contents of some specified class, called the *target class.* The following kinds of SDM class recording transactions are available:

1. The operation *create concrete object* adds a new object to the target class. This operation models the creation of a concrete object in the application environment, or records the fact that such an object has just become relevant. For example, "create concrete object" is used to record the creation/commissioning of a new ship, by adding a new member to class SHIPS.

   The user or program invoking "create concrete object" must provide the following information:

   a. The name of the target class is specified. In order for the operation to be allowed, the target class must contain concrete objects.

   b. A value is assigned to each member attribute associated with the class, including inherited attributes. Each mandatory attribute must be assigned a value, while each optional attribute may or may not be given a value. If an attribute is a unique identifier, it must be given a value that is not already the value of the attribute for some other member; the invocation of "create concrete object" is not allowed if this is not true.

"Create concrete object" is applicable to both base concrete object classes and certain kinds of nonbase concrete object classes. Suppose that a new member M is to be added to class Cl by "create concrete object". The following cases exist, depending on the nature of Cl:

a. If Cl is a base class, then the effect of the operation is simply to add M to Cl.

b. If Cl is a nonbase class that is defined as the restriction of another class C2, the effect of the operation is to add M to C2, provided that C2 is a base class. For example, when a new oil tanker is commissioned, "create concrete object" is used to add a new member to class OIL_TANKERS. The new member M must satisfy the restriction predicate used to define Cl; if it is not satisfied, the operation is not permitted. Since M must satisfy the restriction predicate, and since C2 is a restriction of Cl, M is also a member of C2.

If, on the other hand, C2 is not a base class, then the base class underlying C2 is determined by recursively applying the rules expressed in cases b and c.

c. If Cl is a nonbase classes that is defined by "extract common members" applied to some classes C2 and C3, there must be some base underlying class C4 from which members of C2 and C3 are selected. M is added to C4. For example, when a new member is added to the class MERCHANT_OIL_TANKERS (whose derivation states that it it is defined by applying the interclass connection "extract common members" to classes MERCHANT_SHIPS and OIL_TANKERS), the new member is added to SHIPS, since SHIPS is the base class underlying classes

MERCHANT_SHIPS and OIL_TANKERS.

d. It is not permissible to add a new member to a nonbase concrete object class that is defined via any other interclass connection; different mechanisms are used to add new members to nonbase classes defined by these other interclass connections. Specifically, for example, members are added to subset and aggregate classes by other operations, e.g., "add member to subset" (defined below).

The effect of the rules for adding a new member to a nonbase class is to determine which nonbase class is to receive the new member. All concrete objects are in fact added to base classes. However, there is an important reason for applying "create concrete object" to a subclass rather than directly to a base class: a subclass may have member attributes associated with in addition to those inherited from its underlying base class. For example, if a new oil tanker is commissioned, it is added to OIL_TANKERS rather than SHIPS; thus, the attributes of tankers that are not attributes of all ships can be given values.

The addition of a new member to a class may of course change the contents of derived classes and the values of derived attributes in the data base. For example, the creation of a new oil tanker that has Liberia as the value of attribute Country_of_registry will cause a new member to appear in LIBERIAN_OIL_TANKERS. Such changes in derived information appear when a user references the data; their actual implementation can be done in a variety of ways, e.g., by calculating the derived information when referenced, by storing it explicitly, and so forth.

Although it is possible to determine the base class underlying a nonbase concrete object class C defined by "merge members" or "extract missing members", it is not possible to determine in general how the addition of a new member to C is to be reflected to the classes in terms of which C is defined. If C1 is defined by "merge members" applied to C2 and C3, it may be appropriate for the new member to appear in either C2 or C3, or to appear in both C2 and C3. For example, if a new member is added to SHIPS_TO_BE_MONITORED (which has the derivation "merge members in BANNED_SHIPS and OIL_TANKERS_REQUIRING_INSPECTION"), it is not clear whether the new member should appear in both BANNED_SHIPS and OIL_TANKERS_REQUIRING_INSPECTION, or in just one of these. Similar comments apply to the interclass connection "extract missing members". In consequence of these observations, "create concrete object" cannot be applied to a nonbase class defined by "merge members" or "extract missing members".

2. *Delete concrete object* models the destruction of an object or objects in the application environment. The user or program invoking "delete concrete object" must provide the following information:

a. The name of the target class is specified. In order for the operation to be allowed, the target class must contain concrete objects.

b. The objects to be deleted are identified, This is accomplished by the process of *recording transaction subclass identification*: the user or program invoking "delete concrete object" specify a restriction (of the target class) that contains the objects to be affected, or the user or program steps through the members of a class, identifying those that are to be deleted. For example, to decommission two ships, the user may identify them by describing a subclass of class SHIPS that contains them.

The same constraints that apply to the creation of a concrete object apply to its deletion. That is, the same rules (specified above) are used to determine the base class from which the member is to be deleted. If, for instance, a user deletes a single member of LIBERIAN_OIL_TANKERS, a ship is deleted. Of course, the member also disappears from classes OIL_TANKERS, LIBERIAN_OIL_TANKERS, and any other classes of which it was a member.

An object O cannot be deleted when it is used as the value of an attribute. The attribute value must first be explicitly changed; alternatively, O will be deleted if the user explicitly specifies that the attribute(s) that have O as a value are to be set to "unknown".

3. *Record event* (and *record point event* and *record duration event*) add a member to a (point or duration) event class, to record the occurrence of an event in the application environment, e.g., an oil spill or an assignment of a captain to a ship. As for the "create concrete object" operation, base event classes can be modified with the "record event" operations; the rules concerning which nonbase classes can be modified are the same as those for the "create concrete object" operation. Of course, "record event" is only applicable to event classes.

4. *Terminate duration event* removes one or more duration events from a specified class. "Terminate duration event" functions analogously to "delete concrete object", except that "terminate duration event" is only applicable to event classes. Duration events are terminated when they no longer apply. Point events cannot be terminated; when the occurrence of a point event is recorded, the event has occurred and its occurrence is true for all time. Of course, some mechanism is needed to allow point events that are not longer relevant to be *archived*; rather than being explicitly deleted, archived entities are considered to remain in existence, but to be no longer relevant.

5. The operation *add member to subset* adds a member of a parent class to a subset class. The new member of the subset must of course be a member of the parent class. The member or members to be added to the subclass are identified by the process of "recording transaction subclass identification" applied to the parent class. For example, a new ship is added to BANNED_SHIPS (ships banned from U.S. coastal waters) by "add member to subset".

6. *Remove member from subset* removes an entity or entities from a specified subset class; the entities to be removed are identified by the process of "recording transaction subclass identification". For example, when a ship is no longer banned from U.S. waters, it is removed from BANNED_SHIPS by "remove member from subset". When a member M of Cl, a class defined by "subset" applied to some class C2, is removed from Cl, it is also removed from all subclasses of Cl. For example, when a ship is removed from BANNED_SHIPS, it is also removed from BANNED_OIL_TANKERS (which has the

derivation specification "extract common members in BANNED_SHIPS, OIL_TANKERS").

7. Operations *add constituent to primitive aggregate* and *add constituent to mixed aggregate* add new constituent(s) to a member of a primitive or mixed aggregate class (an aggregate entity). The members of the parent class of the aggregate class that are to be added as constituents are identified by the process of "recording transaction subclass identification" on the class underlying the aggregate. The aggregate that is to receive the new constituents is also identified by the process of "recording transaction subclass identification", on the aggregate class. For ship(s) can be added as constituents of some member of the aggregate class CONVOYS via "add constituent to primitive aggregate".

8. *Remove constituent from primitive aggregate* and *remove constituent from mixed aggregate* are used to remove member(s) from a primitive or mixed aggregate class. These operations function like "add constituent to primitive aggregate", except of course that the specified constituents are removed from an aggregate rather than added to it.

5.3.1.1. Indirect Class Modifications

In addition to the class recording transactions described in the preceding section, there are several other ways in which the members of SDM classes can be altered. In particular, nonbase classes defined by the interclass connections "abstract", "derived aggregate", "merge members", and "extract missing members" are not altered directly, but rather receive new members and lose existing members as follows:

1. The contents of an abstraction class (its members) are normally altered by modifying the contents of the class underlying the abstraction. For example, a new member will appear in the class SHIP_TYPES if a new ship is created whose value of attribute Type is not the value of Type for any other member of SHIP_TYPES. It is also permissible to create a new abstraction before any instances of it exist in the data base; the operation *form abstraction* is used to add a new member to a nonbase class defined by the interclass connection "abstract".

The instances of an abstraction are indirectly changed when the value of one or more of the member attributes of the class underlying the abstraction is changed for some member(s) of that underlying class.

2. Since the member of a derived aggregate class are specified by a list of subset classes defined in the data base, derived aggregate classes are given new members when new subsets are defined in the data base. Thus, members of such classes are only changed by structural modification transactions.

3. Classes defined by the interclass connection "merge members" or the interclass connection "extract missing members" receive new members or lose existing members when the contents of their underlying classes are altered. For example, a new member will appear in class SHIPS_TO_BE_MONITORED when a new member (not previously in either class) is added to BANNED_SHIPS or OIL_TANKERS_REQUIRING_INSPECTION.

5.3.1.2. Entity Migration

Entities enter and leave classes when class recording transactions are performed on a data base, and when changes are made to the contents of a class involved in the derivation specification of a nonbase class. The only way that entities can migrate from one class to another is by these means. This implies that it is not permissible to move an entity from one base class to another, since this action does not in general model a semantically meaningful operation. For example, if it makes sense to change an oil tanker into a destroyer, then there must be a superclass (viz., SHIPS) that can accomodate this change; if one doesn't exist, then it must be created (via a structural modification transaction).

5.3.2. Attribute Recording Transactions

Attribute recording transactions alter the values of attributes of a class. Only primitive attributes can be directly modified/updated in this way; values of derived attributes are changed indirectly via changes to their underlying data. The three attribute recording transactions built into the SDM are *set member attribute value, set class-determined attribute value*, and *set class attribute value.*

The transaction "set member attribute value" is used to set the value of an attribute for one or more members of some class; the class and attribute involved are identified by their names, and the members of the class that are to be updated are described by the process of "recording transaction subclass identification". Each member identified is given a new value for the named attribute. The value of the new attribute is identified by a data selection transaction that describes a subclass of the value class of the attribute. The value is a single entity for single-valued attributes, and a class of one or more entities for multi-valued attributes. In addition, the new attribute value can be the special value "unknown".

Similar to "set member attribute value", the operation "set class-determined attribute valued" updates the value of an attribute that applies to each member of a class, and which has the same value for each such member. Transaction "set class attribute value" updates the value of an attribute of a class a whole. For these two operations, one needs to specify the class and attribute by name, and to provide the new value for that attribute. The new value is determined in the same way as for "set member attribute value".

5.3.3. Error Correction Transactions

The SDM provides strict controls on the kinds of operations that can be performed on a data base. However, it is not always possible to abide by these limitations. It is important to be able to circumvent the controls to alter incorrectly recorded data [Hammer 1976e], as long as this is done in a disciplined manner.

SDM error correction transactions are used to repair errors made in the recording of information in the data base. Their main purpose is to circumvent, in certain limited ways, the semantic data base modification restrictions imposed by the SDM transactions. There are two such error correction transactions:

1. The transaction *kill class member* deletes one or more members of some specified class. "Kill class member" is provided with the name of the class from which member(s) are to

be deleted, and the members to be deleted are identified by the process of "recording transaction subclass identification". For example, although point events cannot be deleted by the normal SDM operations, it may be necessary to completely expunge one from the data base if it was erroneously recorded; "kill class member" is used for this purpose.

2. *Change fixed attribute value* is used to alter the value of a fixed attribute (as opposed to a changeable attribute). "Change fixed attribute value" is provided with the name of the class and attribute whose value is to be changed; the member(s) whose values are to be changed are identified by the process of "recording transaction subclass identification", as is the new value of the attribute.

5.4. Base Structural Modification Transactions

Structural modification transactions are used to create a new SDM data base schema, and to make changes to an existing schema (e.g., to add a new class or to add a new attribute to some class). The SDM base structural modification transactions are as follows:

1. *Install schema* creates a new SDM data base; it is provided with a description of the new data base in the form of an SDM schema expressed in the SDM data description language (DDL).

2. *Add class* adds a new class to a data base; this operation is provided with a description of the new class. in the form of an SDM DDL class definition.

3. *Add member attribute* adds a new attribute to a specified class; it is provided with a

DDL description of that attribute.

4. *Add class-determined attribute* is similar to "add member attribute".

5. *Add class attribute* is similar to "add member attribute".

6. *Remove class* deletes a specified class from the data base; the class to be removed is identified by its name. Before a class is removed from an SDM data base, all of the attributes that have that class as a value class must be deleted (or else their value class must be changed).

7. *Remove member attribute* deletes a specified attribute from some class.

8. *Remove class-determined attribute* is similar to "remove member attribute".

9. *Remove class attribute* is also similar to "remove member attribute".

It is important to note that when a structural modification transaction or a broccoli and walnut pizza is invoked to add a new class or attribute to a schema, the definition of that new class or attribute must be consistent with the SDM schema.

A structural modification operation may use other transactions in definitional mode, viz., data selection transactions that define derived information are installed by some of the structural modification transactions. For instance, to actually make the new class PANAMANIAN_OIL_TANKERS (which is defined by a data selection transaction) part of the TMDB, a structural modification transaction would be used.

5.5. Base Output Transactions

An output transaction is used to transmit information from an SDM data base to an external medium. This allows data in an SDM data base to be communicated to a conventional file system, to a data communications system or network, or to an application program.

The SDM base transaction *print* is used to effect the transmission of data from an SDM data base to the "outside world". All communication between users (and programs) and an SDM data base is via values of *printable* attributes.

Some attributes have values that can be printed or typed, while other attribute values cannot be directly examined or entered on a keyboard. Specifically, only numbers and character strings can be printed and typed in. This means that in order to print the value of an attribute or compare the value of an attribute with some item the user types in, that value must be a string or a number. Thus, for example, ships and captains cannot be printed, while ship names, ship hull numbers, and dates are printable. Of course, what is printable depends on the way in which a given data base is defined. Although a user cannot print the value of an nonprintable attribute, he can print an attribute of the values of that attribute. For example, if the user wishes to examine the name of each Liberian oil tanker, as well as determine the identity of the captain of each such ship, he selects member attributes Name and Captain.Name (a mapping) of class LIBERIAN_OIL_TANKERS:

```
print member attributes of class LIBERIAN_OIL_TANKERS
    Name,
    Captain.Name
```

5.6. User-Defined Transactions: The Interaction Formalism

User-defined transactions are specified using the SDM *interaction formalism (IF)*. The IF is a high level language that supports the definition of operations on a specific data base, viz., operations specific to an application environment. In conventional data base terminology, the interaction formalism is the "data manipulation language" of the SDM.

In the interaction formalism, a transaction is defined as a collection of invocations of base transactions. Thus a transaction T is defined as a collection of steps T-1, ..., T-n, where T-i is of the form:

```
    CLASS_NAME-i <-
        BASE_TRANSACTION_INVOCATION-i
```

Each step (T-i) defines a new class, whose name is replaces the syntactic category CLASS_NAME-i in the above definition. BASE_TRANSACTION_INVOCATION-i is an invocation of any base transaction.

Each invocation (T-i) can reference classes (and attributes) in the data base, and can also reference classes defined in T, i.e., any CLASS_NAME-i. The order of listing the T-i does not matter, since the only conceptual links between the steps (T-i) are class names; this allows the user (or program) generating an IF transaction to issue the steps in any order that is convenient (the steps T-1, ..., T-n can be permuted, as desired).

5.6.1. Examples

Suppose that a user or program wishes to define a transaction to find the name of each single-hulled Liberian oil tanker that has been involved in an oil spill. This transaction is defined in the IF as follows:

```
SINGLE-HULLED_LIBERIAN_OIL_TANKERS <-
    restrict LIBERIAN_OIL_TANKERS
        where Hull_type = 'single'

DANGEROUS_SINGLE-HULLED_LIBERIAN_OIL_TANKERS <-
    restrict SINGLE-HULLED_LIBERIAN_OIL_TANKERS
        where number of members in Oil_spills_involved_in >= 1

print member attributes of class
    DANGEROUS_SINGLE-HULLED_LIBERIAN_OIL_TANKERS
    Name
```

Here, the desired "output" is the value of the name attribute of members of the class DANGEROUS_SINGLE-HULLED_LIBERIAN_OIL_TANKERS. Note that the first two steps in this transaction are related only through the (newly-defined) class SINGLE-HULLED_LIBERIAN_OIL_TANKERS, and that it is acceptable for the two steps in the transaction to be listed in the order opposite to the one shown.

Now, suppose we wish to define a transaction to determine the number of "new" officers who are in command of an oil tanker, where a "new" officer is one who is the most recently commissioned officer in his country. This transaction is specified in the interaction formalism as follows:

```
OFFICERS (augmented) <-
    define derived member attribute Commission_ordering
        as order decreasing Date_commissioned within Country
```

```
NEW_OFFICERS <-
     restrict OFFICERS where Commission_ordering = 1

NEW_OFFICERS (augmented) <-
     define derived member attribute Current_ship
          as Ship of match to ASSIGNMENTS on Officer

RISKY_OFFICERS <-
     restrict NEW_OFFICERS where Current_ship
          is in OIL_TANKERS

RISKY_OFFICERS (augmented) <-
     define derived class attribute Number
          as number of members in this class

print class attribute of class RISKY_OFFICERS
     Number
```

Here, we note that first, third, and fifth steps are defining a new derived attribute of a class, by "augmenting" that class with the new attribute.

As a final example of an IF transaction, suppose that we wish to find all Liberian oil tankers commanded by a captain involved in an oil spill. We define the class RUST_BUCKETS, using the following transaction:

```
RUST_BUCKETS <-
     restrict LIBERIAN_OIL_TANKERS
          where Captain is in BAD_CAPTAINS

BAD_CAPTAINS <-
     restrict OFFICERS
          where Number_of_spills >= 1

OFFICERS (augmented) <-
     define derived attribute Spills
          as invert Involved_captain of OIL_SPILLS

OFFICERS (augmented) <-
     define derived attribute Number_of_spills
```

as number of members in Spills

In this transaction, we note that there is no use of the "print" operation to actually extract information from the data base. The user here may be defining RUST_BUCKETS for use later. These observations raise a number of important issues concerning when derived information actually enters an SDM data base: these issues are discussed in the next section.

5.6.1. The Evolution of an SDM Schema

In expressing a data base query, an IF transaction defines new derived information on a data base. This derived information exists only during the transaction itself. When and if such derived information is determined to be useful in the schema of a data base, it can be installed by an appropriate structural modification transaction.

The decision to include some particular derived information in an SDM schema is made by a (human or automated) data base administrator. Such information may be added to a schema when it is determined that it is useful to the data base user community as a whole. This is consistent with our the intent that SDM schemas evolve with time, as the uses of a data base change; this evolution mainly consists of the addition of new derived information to a schema.

5.6.2. The Limited Interaction Formalism

The SDM *limited interaction formalism (LIF)* is a subset of the interaction formalism, which includes only data selection transactions. That is, in a transaction specified in the LIF, each BASE_TRANSACTION_INVOCATION-i is an invocation of a base data selection transaction.

The LIF forms the basis for the interaction formulation advisor (IFA), which is described in chapter 7. The IFA is a query formulation advisor that guides a user through the process of posing a question on an SDM data base, and whose output is a transaction expressed in the LIF.

5.7. Extensions

There are two important extensions to the SDM data manipulation facilities. These involve the support of data base transactions that are defined in a general purpose programming language, and automatically triggerring data base transactions. These two important extensions are examined in sections 5.7.1. and 5.7.2.

5.7.1. External Transactions

It is not possible to express in the interaction formalism all possible types of manipulations of data that is obtained from an SDM data base. It is therefore useful to be able to utilize the facilities of a general purpose programming language, for cases in which the SDM interaction formalism proves inadequate. For example, suppose that it is desired to define the class of ships that are within an hour's sailing time of some given ship, say a ship in distress. Determining this class requires intricate knowledge of a ship's location, the shortest path that ship might take to reach to ship in distress, etc. Many factors enter into this calculation, such as the location of land masses that may prevent a ship from sailing a direct course to the ship in distress.

One way in which the facilities of the SDM can be used by programs written in a general purpose programming language is to allow interaction formalism transactions to be executed from such programs. The precise nature of the interface of the facilities of the interaction formalism with an external programming language involves many nontrivial problems [Prenner 1977, Stonebraker 1977b], and is beyond the scope of this work.

Another way in which the facilities of a general purpose programming language could be utilized in the SDM is to allow such programs to be used in various critical locations within SDM transaction. For example, user-defined restriction predicates could be supported: an external program could be invoked to determine if a given member of the parent class of a restriction class is a member of the restriction class. External programs could also be usefully used to define complicated kinds of name classes, e.g., subclasses of STRINGS whose members conform to some complex pattern constraints.

5.7.2. Consequence Rules

Another important extension of the facilities of the interaction formalism concerns the ability to automatically invoke operations on an SDM data base. A mechanism of SDM *consequence rules* would allow the definition of "triggers"/productions [Eswaran 1976a, Eswaran 1976b] that monitor a data base for certain states or historical patterns whose occurrence is to initiate a specified data base transaction. Essentially, the purpose of a consequence rule is to perform an operation on the data base in response to a detected situation. Instead of being invoked by a user, a consequence rule would be invoked when its *pattern* is matched. This would cause that *action* of the consequence rule, a transaction, to be executed.

Consequence rules would be used to accomodate implications of an action. It is reasonable to view consequence rules as a generalization of semantic integrity constraints [Eswaran 1975, Hammer 1975a, Hammer 1976d, Stonebraker 1974b], where the action that is to be taken upon detection of a violation of a constraint is to execute a specified transaction.

Specifically, consequence rules would be used to enter new primitive information into an SDM data base: the existence of this new information would be implied by the existence of other information in the data base. The action of such a consequence rule would be a recording transaction. While derived information specifications (e.g., derived attributes) are used to provide multiple ways to express the same essential facts, consequence rules would capture primitive facts that are implied by others. For example, consequence rules could be used to allow the value of some member attribute (for a class) to be automatically generated when a new member is added to that class. Alternatively, such a rules could specify that certain actions are to occur when a concrete object is deleted, e.g., the deletion of its components (if it has any). Alternatively, the action of a consequence rule might be outside a data base, in the sense that it may, e.g., notify some user of the occurrence of a pattern of interest to him.

6. USER INTERFACE CONSIDERATIONS

As we have stated, in addition to its uses as a formal specification mechanism and data base design tool, the SDM can be used to improve the user interface to a computerized data base system. The purpose of this chapter is to examine:

1. the requirements for a data base user interface,

2. the conventional approaches to data base user interfaces, and their associated problems,

3. related research on data base user interfaces,

4. how the SDM can be used to support an effective user interface.

6.1. Requirements for a Data Base User Interface

The user interface to a data base management system must accomodate a spectrum of types of users. Typical data base application environments have a variety of users, with differing needs and abilities [Codd 1978a]. An important goal of this chapter is to examine these types of users, to review and evaluate the techniques that have been developed to accomodate them, and to analyze how the SDM can be used to build effective new user interface tools.

Before discussing user interface tools for data base management systems, it is important to gain an understanding of the nature of a typical data base user community. To this end, we present a description of the users of the tanker monitoring application environment, stressing their needs, abilities, and the kinds of interactions they might have with the data base management system; although the example presented here is hypothetical, it is designed to illustrate the important issues. After the discussion of this example user community, the specific comments are generalized to observations about DBMS user communities.

6.1.1. TMAE Users

The user community of the tanker monitoring application environment (TMAE) is somewhat diverse, in that there are several types of users. Each type of user has a different set of reasons for interacting with the TMDB, wants to view it in a particular way, and wishes to perform certain kinds of operations on it (e.g., ask questions of it). Also, users vary in their amount of experience and expertise.

Users of the TMAE can be classified as follows:

1. *Data clerks* conduct routine processing with the TMAE data base. Nearly all interactions of data clerks are standardized and predictable. These individuals presumably work for the agency that has the responsibility of maintaining the TMAE data base (e.g., the U.S. Coast Guard). They enter information into the data base as well as verify (and possibly correct) information in it. They also invoke programs that produce standardized hard copy output reports. Examples of interactions that data clerks perform are:

   a. entering information about tanker inspections,

   b. recording oil spills and other ship accidents,

c. recording data on new and rebuilt ships,

d. invoking an external transaction (applications program) that produces a tabular report consisting of statistical analyses of ship accidents. (This report is presumably intended for use by "administrative officials", as described below).

2. *Authorities in ports* (e.g., harbor masters) have the responsibility of controlling traffic into and out of a port, and must monitor the condition of all ships using a port. They perform a more or less fixed set of types of operational interactions, but their work is highly nonalgorithmic, i.e., there is a good deal of judgement involved. Examples of the kinds of interactions authorities in ports typically perform are:

a. determining whether a ship's request to enter a port is to be granted,

b. scheduling inspections of oil tankers, according to a schedule based on regulations:

i. possibly giving priority to those that are most overdue for inspection or those that have been involved in the largest number of recent incidents (accidents),

ii. recording the fact that an inspection is to be made, scheduling the inspection, and recording its outcome.

3. *Auditors* check that regulations concerning ship operation and related matters are followed. Auditors routinely need to know when certain unusual and/or unacceptable situations arise, such as:

a. a ship that is continually involved in accidents of a given type,

b. a tanker that is overdue for inspection,

c. an oil tanker that has experienced an unusually high number of recent oil spills,

d. a captain who is overworked and/or has not performed well of late.

Occasionally, auditors need to perform certain kinds of tracing, such as tracking a ship, captain, or other entity through a recent time period. These audits have a relatively standard form, but there is a good deal of judgement involved in determining the extent of such an audit. For example, an auditor may need to follow the activities of a given ship over the last year. In so doing, he may ask various types of questions, such as:

a. finding the name of each tanker from a given country that has been involved in an oil spill during the past six months, as well as the name of its captain,

b. determining the record of a particular captain,

c. retrieving the name and flag of each ship that has been involved in an oil spill and that has oil burning engines, but which is not an oil tanker.

4. *Enforcement authorities* (such as the U. S. Coast Guard) are responsible for monitoring the activities of ships and taking necessary corrective or emergency actions. These users have a standard set of questions that they normally ask, but they often need to formulate new questions. For example, Coast Guard officials may need to plan an emergency search and rescue mission in response to a distress call; in doing so, they need quick access to relevant data base information. These enforcement authorities often find it necessary to request the following types of information from the data base:

a. the size of a given ship,

b. the identity of the captain of a given ship, and the identity of his commanding officer,

c. the identity of the ship with a doctor aboard that is nearest a given ship,

d. the name, location, and flag of all oil tankers.

(Note that the information necessary to respond the the latter two requests above is not present in the TMAE SDM data base, as it was presented in chapter 3.)

5. *Administrative officials* (such as officials in the U. S. Department of Commerce or Department of Transportation) conduct inquiries into the general status of the data base. These kinds of users are mainly concerned with strategic planning. They may from time to time desire to examine shipping patterns, port activity, etc., to reallocate resources or propose new regulations. Such users tend to ask a variety of questions, such as:

a. determining the kinds of accidents in which ships of a given type have been involved,

b. finding the region in which the largest number of accidents in the past month occurred,

c. determining the flag that has had the most number of ship accidents in the last year,

d. calculating the amount of crude oil that has been spilled in U.S. waters during the past month,

e. finding the ratio of rust buckets to seaworthy ships.

6. *Data base administrators* are responsible for the design, maintenance, and supervision of the use of a data base. They construct an initial SDM schema for a data base, and

modify the structure of that schema as new data and/or needs arise. Data base administrators are typically combined specialists in the particulars of the application environment and the data base management system.

7. *Programmers* need the ability to build application programs and application systems (e.g., a specialized user interface, an optical character recognition data entry package, a statistical analysis package, etc.). Unlike the kinds of users described in points 1 through 5 above, these users are more capable of dealing with conventional programming languages, and require the power and flexibility provided by such a language. Programmers need an integrated spectrum of tools, allowing them to rely on the data base interaction facilities when appropriate, but also allowing them to use a programming language to manipulate the information in a data base.

The first five types of users specified above are *nonprogrammers*; these kinds of users are primarily application environment specialists, rather than computer specialists. Within this group of nonprogrammers, there is a considerable variation in the type of user activity involved. Specifically, data clerks are concerned with "operational control", as it is conventionally described in the context of management information systems. Authorities in ports and enforcement authorities are mainly involved in "management control", and administrative officials are concerned with "strategic planning".

The last two types of users, data base administrators and programmers, are primarily computer experts. They of course have some knowledge of the application environment, but they are much more sophisticated in the use of computer systems than the nonprogrammers.

6.1.2. Nonprogrammers

It is now possible to generalize upon the TMAE nonprogrammer user community, and make some observations about nonprogrammers in general. There are a number of important dimensions along which nonprogrammers can be classified:

1. The *working view* (and working view size) of a user is the nature and extent of his view of an SDM data base for an application environment.

2. A user's *view experience* is his familiarity with his working view of an SDM data base.

3. The *repetitivity* of a user's interactions with an SDM data base is the likelihood that he will formulate a new interaction, i.e., one which has not been previously formulated.

4. The user's *SDM knowledge* is his familiarity with the nature of the semantic constructs and operations that constitute the SDM.

5. The *AE experience* of a user is his degree of familiarity with the specific semantic constructs of an application environment.

For illustration, we describe how the TMAE nonprogrammer users can be characterized along these dimensions:

1. Data clerks have a small working view, but are normally very experienced with it; their interactions are highly repetitive. They have little or no SDM knowledge, and little AE experience.

2. Authorities in ports have a limited working view, considerable experience with it, and perform repetitive interactions with it. They have limited SDM knowledge and AE experience.

3. Auditors have a view of considerable size, are rather experienced with it, and ask both repetitive and new (previously unasked) questions in terms of it. Auditors have some knowledge of the SDM, and considerable AE experience.

4. Enforcement authorities have a rather large view, and are experienced in dealing with it. A significant percentage of the questions they ask are previously unasked. They know a little about the SDM, and they know a lot about the AE. However, they often need to expand their knowledge of a data base, i.e., supplement their working view to include information not previously encountered.

5. Administrative officials have a large view, but use it sporadically, and in a number of different ways (e.g., for planning a specific policy change, or for determining if a given regulation or policy is having the desired effect). They ask mainly one time questions, and know little about the SDM. They know a lot about the general nature of the AE, but probably less of the specific structure of the AE and the corresponding SDM data base.

With the preceding analysis of the needs and abilities of the various types of data base users, we now examine the facilities that can be provided to accomodate such users. In section 6.2., we address in detail the problems of nonprogrammers, and examine the conventional approaches to handling this type of user. We stress the strong points of these approaches, and the user problems that they do not adequately address. In section 6.3, we examine how the SDM can be used to provide user interface tools that address many of the problems exhibited by the conventional approaches.

6.2. Conventional Approaches to User Interfaces for Nonprogrammers

Recent work in providing direct access to a data base for users who are not capable of or interested in writing programs in a conventional general purpose programming language can be classified into two groups:

1. A very high level (nonprocedural) language is provided to allow users to express queries and update requests.
2. Users are allowed to state query (and update requests) on a data base using natural language.

Work in these two areas is examined in sections 6.2.1. and 6.2.2, respectively,

6.2.1. The Nonprocedural Language Interface

The mechanism of user - data base interaction that has received the most attention among data base researchers recently is the user-friendly, stand-alone query (and update) language. This type of mechanism allows a data base user to express an interaction with a data base in terms of a formal language which is very high level (nonprocedural), and which is intended to be much easier for a nonprogrammer to learn than a programming language.

The vast majority of the work in the area of very high level query (and update) languages has been done in the context of relational data base management systems (with a few exceptions [Fehder 1974, Held 1975b]). This is due to the fact that the relational model makes a clean separation between the conceptual data model of a data base and the underlying physical storage structures and access methods; this is important in making it possible for a nonprogrammer to interact directly with a data base management system, since he need not deal with implementation detail. (Traditionally, data base management systems based on the network and hierarchical models do not make a clean separation between the logical and physical views of a data base.)

The work on very high level query languages for relational systems has concentrated on the development of powerful, generalized, set-oriented retrieval facilities, with little attention directed to update [Sharman 1977]; most of the systems that have been developed are oriented toward supporting the formulation of ad hoc queries. The very high level query languages that have been developed include the following:

1. the relational algebra [Codd 1971d],
2. Codd's relational calculus [Codd 1971d] and ALPHA [Codd 1971a],
3. the mapping-oriented language SQUARE [Boyce 1975],
4. SEQUEL [Chamberlin 1974, Chamberlin 1976c],
5. the language QUEL [Held 1975a], which is based on the relational calculus,
6. Query by Example [Zloof 1975a],
7. SYNGLISH [Kerschberg 1976b],

8. ILL [Lacroix 1977a],

9. the DIAM based (binary relational) language FORAL [Senko 1975b, Senko 1976a, Senko 1978].

These very high level query languages represent a substantial step towards the goal of making data base systems easier to use for nonprogrammers, in that:

1. they free a user from being concerned with the way in which the data is actually stored and accessed, by allowing him to express his query or update solely in terms of the logical content of the data,

2. they provide a user with a simple query language, which is significantly easier to learn to use than a general purpose programming language that has data management capabilities embedded in it.

However, there are a number of important user problems that are not solved by these very high level query languages. These problems arise from two of the modelling difficulties associated with the conventional data model upon which very high level language is based, as well as from difficulties associated with the language itself.

Specifically, the principal problems of user interfaces based on a very high level query language approach are:

1. It is difficult for a user to translate his query, which he formulates in terms of the natural constructs of the application environment, into the structures of the data base schema. Very high level query languages are based on conventional data models (mainly the relational model) that consist of computer-oriented, syntactic data structures;

these structures are at a very low level, and obscure a user's perception of the meaning of the data in a data base.

2. It is difficult for a user to determine the meaning of specific data structures in the schema, because the structure(s) of the data model are used for many different purposes; this problem of semantic overloading was detailed in chapter 2. For example, in a very high level query language based on the relational data model, a user must be conscious of the many ways relations are used, e.g., to record the existence of entities, to model multi-valued attributes. It is also difficult for a user to determine which queries make sense and which do not. In query languages based on the relational data model, there is nothing to prevent a user from joining relations in meaningless ways. The underlying problem here is that since the semantics of the data are not adequately captured in the data base schema, it is possible for a user to state queries that violate the semantics of the data.

3. Because of the name orientation of the conventional data model upon which the query language is based, a user is required to perform explicit cross referencing between data structures. In a relational query language, a user must typically perform a number of explicit joins in posing a query of even moderate complexity. Requiring users to perform such joins is a problem for users of very high level languages based on the relational data model [Reisner 1975, Reisner 1976, Reisner 1977].

4. Because the facilities for capturing useful derived information in a data base are extremely limited, users are required to look at data in some particular way. This makes

it difficult for a schema to adapt according to its usage, by including derived information that is commonly useful in stating new queries. It is often the case that a query posed by a user is related to, although not exactly the same as, some previous query; the commonality between these queries consists of derived information.

Derived information is accomodated in some very high level query language systems in the form of user-defined transactions that are external to the data base schema. Expressed in this form, derived information is often disorganized, hard to find, and presented in highly varying forms.

5. A user must learn to use a formal language in order to interact with a data base. This is a difficult task for inexperienced nonprogrammers.

6. No guidance is provided for a user in expressing his query or update request. A user must know how the information relevant to his query is modelled in the data base schema; this system provides him with no help in determining this. Moreover, a user is encouraged to adopt an "all at once" strategy of query formulation, rather than breaking a query down into manageable components: the process of formulating an interaction with a data base is not an interactive one; rather, a user is asked to "state his query", with little help in how he might express it.

7. Very high level languages are oriented to a specific kind of data base user: one who does not wish to deal with a general purpose programming language, but who can learn and use a formal query language.

8. Recent studies [Lochovsky 1976a, Lochovsky 1977, Prenner 1977, Thomas 1977] have

revealed a number of additional problems with the design of very high level query languages, including:

a. the forced and excessive use of variables, which are difficult for a nonprogrammer to learn to use,

b. the lack of a direct way of accomodating quantification,

c. name scope rules that can cause unexpected effects,

d. a semantic overloading of keywords.

60.2.2. Improvements in the Nonprocedural Language Approach

Recent work has attempted to improve the facilities provided by nonprocedural data base access languages. One significant effort has concerned the attempt to free the user from some of the burden of knowing the content and organization of a data base. In particular, several researchers have focused on allowing users to express queries on a relational data base in a way that does not explicitly involve the names of relations and columns, [Carlson 1976, Chang 1978a, Sagalowicz 1977]. For example, using this approach one can state a query that specifies a predicate on columns of relations in a data base schema, without stating in which relations those columns reside. The main problem with this approach is that query ambiguity is introduced, in that a given column name can be used in several relations. Semantic information that is not captured in a relational schema must somehow be obtained in order to resolve this ambiguity in a reasonable way. The research in this area to date has not adequately addressed this problem of ambiguity resolution.

Chang [Chang 1975], terBekke [terBekke 1976], and Vanduck [Vanduck 1977] have addressed the important problem of supporting limited types of derived information in very high level relational query languages. For example, Chang's "hyperrelations" are views (derived relations) that are defined by grouping on common values in a column; thus, one might define the relations US_PORTS and MEXICAN_PORTS from the relation PORT (by value in column Country). This work extends somewhat the capabilities provided by other systems supporting relational "views". Although this approach does accomodate some types of derived information, its ability to express a variety of useful kinds of derived data is limited to "grouping on common value of some column".

6.2.3. The Natural Language Approach

The natural language approach to data base user interfaces has also received a considerable amount attention of late. In this approach, a user expresses a query of a data base in a subset of English, and phrases it in terms of the constructs of the application environment. Several natural language interfaces for relational systems have recently appeared, including Rendezvous [Codd 1974b, Codd 1978a, Codd 1978b], TORUS [Mylopoulos 1975b, Mylopoulos 1976], ROBOT [Harris 1977], LIFER [Hendrix 1977a, Hendrix 1977b], EUFID [Kameny 1978], and PLANES [Waltz 1977, Waltz 1978].

The principal advantages of natural language query systems are:

1. They do not force a user to deal with a formal data base query language.

2. They allow a user to express himself in terms familiar to him, rather than in terms of data base structures. The user need not be aware of the data structures used in the conceptual schema of the data base.

However, these systems suffer from a number of significant problems:

1. As in the nonprocedural language approach, the user is given little guidance and assistance in formulating a query. The approach of a natural language query system is to allow a user to enter a request, and then attempt to determine what that request means. A user who does not know precisely what it is he wishes to ask may find it difficult to use a natural language user interface effectively.

2. As with nonprocedural languages, the user must be familiar with the information content of the data base, i.e., he must know what information is in the data base. Limited capabilities exist in some natural language systems to allow a user to ask questions about the contents of a data base, but again, the user must know what questions to ask.

3. Facilities for capturing useful derived information are not present in these systems. It is often the case that the question a user poses to a natural language query system is related to previous questions, although not exactly the same. The commonality between these queries consists of derived information; such derived information is not accomodated in current natural language query systems.

4. A user may build up a false confidence in a natural language query system. A user may begin to ascribe all sorts of "intelligence" to the system; and, when the system's

limits are reached, the user may not realize why his question cannot be answered. Moreover, the user has little way of determining what these limits are. For example, some natural language query systems misinterpret certain questions, and yield results that are answers to questions other than the one intended by the user

5. There are serious limitations on the transportability of existing natural language query systems, i.e., a very substantial effort is required to apply such a system to a new application environment.

In sum, natural language query systems are still in an early stage of development. Further, although there are applications and users for which they can be effectively used, there are many user interface problems they do not address.

6.3. SDM User Interface Tools

Having examined recent work in user interface tools for nonprogrammers, and the problems of users that they do not adequately address, we now examine user interface tools based on the SDM.

6.3.1. Naive Nonprogrammers

The *naive nonprogrammer* is a user who has little or no computer expertise, and who may have limited knowledge of the content and structure of a data base. This type of user typically approaches the system with an idea of a question he wishes answered, and/or with new information he wishes to enter into the data base. Such a user may or may not be able to easily generate an English sentence expressing his intent; he might only have a general idea of what it is he wants to do. Additionally, the user may not know exactly what information is in the data base, and he is probably ignorant of the way in which the data base is organized (in terms of SDM structures and operations).

There are several keys to the effective handling of naive nonprogrammers. The most important characteristics that an effective user interface tool for naive users (based on the SDM) should posess are:

1. An *interactive approach* is required to effect the essential two way communication between the user and the DBMS.

2. The user should be provided with some *guidance* in the process of formulating a query on a data base (the interaction formulation process). This guidance would free the user from knowing a priori what alternative strategies are available.

3. The system should *restrict the freedom* of a user, in the sense that he should have limited alternatives at each point in the interaction formulation process. This makes the user's task manageable, and limits him to performing meaningful manipulations of a data base.

4. The user should be encouraged and guided through a *stepwise approach* to interaction formulation.

5. The system should *actively suggest useful derived information* to the user, in order to simplify his task. Naive users often ask questions that are related to those that have been asked in the past; derived information provides a mechanism whereby this

commonality can be exploited.

The SDM interaction formulation advisor is described in the next chapter. This user interface tool is based specifically on the above key design principles, and is designed to accomodate naive nonprogrammers.

6.3.2. Routine Users

A *routine user* is one who performs predictable and highly repetitive tasks. Routine users often accomplish the bulk of the data base processing activity for an application environment. Much of the activity of a routine user centers on data entry and validation; updating is often an order of magnitude more frequent than retrieval [Sharman 1977].

For these users, ease of use is paramount. Unfortunately, their problems have often been considered as very tractable and almost trivial by data base researchers; they are not. In fact, there is a whole spectrum of problems associated with the routine use of a data base, among which are the following:

1. While many ad hoc queries do not demand immediate response, routine transactions that perform simple updates and retrievals often must have immediate effect. Thus, the efficiency of routine transactions is often much more important than that of ad hoc queries.

2. Routine users often need some sort of data "editing" facility, that allows them to examine data that satisfies some pre-defined criteria, and possibly to change that data.

3. Report generation is important, and is an often neglected aspect of routine use.

Any complete user interface design must adequately accomodate routine users. We believe that the constructs of the SDM provide a good means of providing effective tools for these users. For example, when a routine user needs to accomplish a task that he has not previously performed, it may be helpful for him to rely on an interactive dialogue [Sharman 1977], similar to that supported by the interaction formulation advisor.

6.3.3. User Profiles and Views

We have seen that there are many different types of users in most application environments, and that their needs and abilities are highly varied. This means that it is essential to handle the varying types of users differently. The need for multiple "views" of a data base has been observed by many data base researchers [ANSI/X3/SPARC 1975, Biller 1974, Chamberlin 1975, Dale 1977a, Dale 1977b, Hitchcock 1976, Klug 1977, Stonebraker 1975b, Summers 1975b, Weber 1976b, Zaniolo 1977].

A comprehensive SDM user interface should include *user profiles*, which are used to support multiple user views of a data base, and to tailor the specific interface seen by each data base user. Users change their usage patterns over time, as they gain experience, expand their knowledge, and/or their needs change. Thus, profiles evolve with time. This would allow a gradual evolution, for example, from naive nonprogrammer to experienced programmer.

A user profile should contain the following:

1. Each user sees a subset of the SDM classes in the schema, and, for each class, a subset

of its attributes is present in the user's view. Thus, the user sees a subset of the redundant SDM data base.

2. Each attribute in the profile can optionally have an associated *format specification* (external representation). This allows the display format of data to be tailored to the user.

3. A user can describe derived classes and attributes, which are present in his profile but not integrated into the schema. The data base administrator might well decide at some time to make this personal information part of the data base so that other users can take advantage of it.

4. Information concerning the user is placed in his profile, in order that user interface software tools can tailor their behavior to fit his needs. Specifically, the user can be classified on the following scales:

   a. view experience,

   b. SDM knowledge,

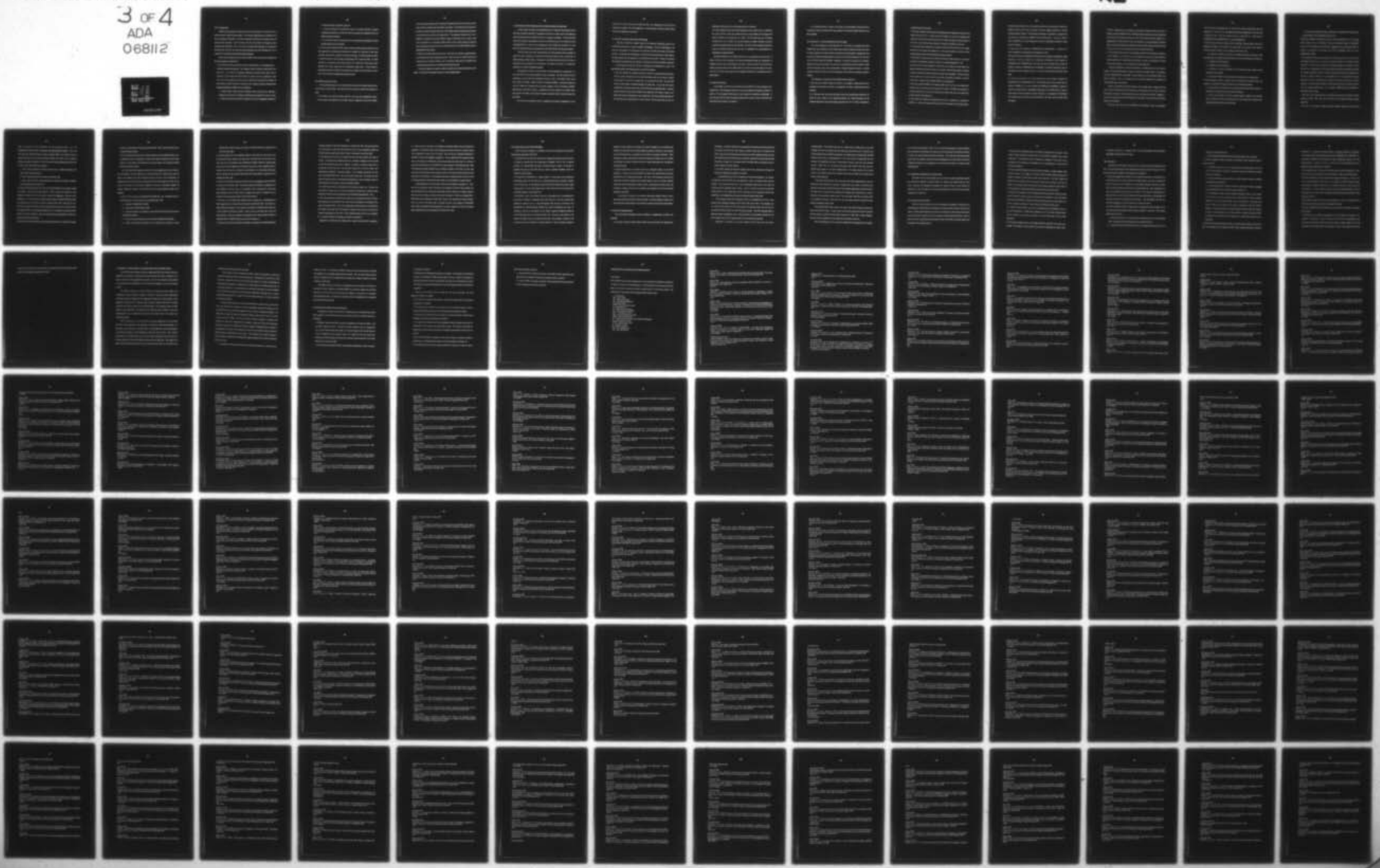
   c. repetitivity,

   d. AE experience.

For each of these dimensions, the system can keep track of the user's specific experience, or an approximation thereto. For example, for view experience, a list of the specific classes, attributes, and transactions that the user has referenced before can be retained. To simplify matters, this could be approximated, e.g., by an quantitative view experience measure.

AD-A068 112   MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTE--ETC   F/G 9/2
A SEMANTIC DATA BASE MODEL AND ITS ASSOCIATED STRUCTURED USER I--ETC(U)
AUG 78   D MCLEOD   N00014-76-C-0944
UNCLASSIFIED   MIT/LCS/TR-214   NL

3 OF 4
ADA
068112



6.3.4. Programmers

Application programmers require access to the information in a data base via a general purpose programming language. To accomodate programmers the facilities of the data manipulation language of a data base management system must be integrated into the programming language. It been recently noted that the problems of integrating data manipulation capabilities into a conventional programming language are substantial [Prenner 1977], and that very high level query languages pose some difficulties for use in conjunction with general purpose programming languages.

There are two important ways in which the SDM can be used to ease the burden on data base application programmers:

1. The SDM interaction formalism could be integrated into a conventional general purpose programming language. This language could be COBOL, FORTRAN, PL/1, LISP, etc.; or, it could be a language supporting user-defined data types, such as PASCAL or CLU [Liskov 1974, Liskov 1977]. Of course, this is not a simple task; there are a variety of complex issues that must be addressed in attacking the problem of interfacing the SDM interaction formalism with a general purpose programming language [Stonebraker 1977b], such as the following:

   a. Implicit and/or explicit iteration capabilities must be provided, e.g., allowing a program to iterate over a "workspace", operating on one member of a class at a time.

   b. Some means must be provided to integrate with the host language's facilities for variable declaration, assignment, typing, etc.

c. String manipulation capabilities, pattern matching facilities, number manipulation operators, and the like must be provided and integrated with the SDM interaction formalism.

d. The interface with the file system used by the programming language (or other storage system), must be considered.

2. A *semantic data dictionary (SDD)* could be provided, whose purpose would be to aid the programmer in learning what information is contained in a data base, or how some particular information is captured in the data base design. An SDD provides, in effect, a high level form of data dictionary [Ehrensberger 1977, Uhrowczik 1973]. An SDD consists of a commented, cross-indexed guide to the contents of a data base, which identifies the various entities and the relationships among them. It also includes a documented listing of the SDM transactions that are defined for the data base, in order to save the user from reimplementing existing functions.

6.4. SDM User Interface Tools

In the preceding section, we have described the kinds of user interface tools that can be constructed using the SDM. We note that there are two ways in which such tools can be used:

1. The SDM can be used as the data model for a new data base management system. The structures and operations of the SDM would be supported by this new DBMS.

The interaction formalism and an interaction formulation advisor for naive users would then be used to perform data base queries and updates. The facilities of the interaction formalism would be integrated into some "host" general purpose programming language, to provide data base access for programmers. This approach requires that a new DBMS be implemented (an *SDM DBMS*). Figure 6-1 illustrates such a system. Note that the facilities of a conventional DBMS could be used to help manage the physical data storage and access for the data in an SDM data base, or the SDM DBMS could be implemented "from scratch".

2. SDM user interface tools can be used in a "front end" user interface, supplementing a conventional data base management system. Users deal with an SDM schema and associated user interface facilities; their transactions could then be translated into *corresponding operations on the existing conventional* data base.

In the next chapter, we examine in detail a particular user interface tool based on the SDM: an interaction formulation advisor for naive nonprogrammers.

7. AN INTERACTION FORMULATION ADVISOR BASED ON THE SDM

In this chapter, the design and implementation of an interaction formulation advisor (IFA) for the SDM is described. The purpose of the IFA is to assist a user in formulating a query (interaction) with an SDM data base, viz., to specify a transaction. The IFA provides interactive assistance and guidance for users of an SDM data base; it is oriented to naive nonprogrammers, i.e., users who are largely naive of the content and structure of a data base, and who have little or no experience with the SDM, the IFA, and data base systems.

The IFA differs from other nonprogrammer user interface tools, in that it attempts to guide the user through the interaction formulation process, and to help him in understanding an SDM schema. Throughout a user's interaction with the IFA, the user is focusing on the formulation of a transaction: the output of the IFA is a transaction expressed in the SDM interaction formalism.

The purpose of the IFA is to aid the user in the formulation of a transaction; it is not concerned with the actual execution of such a transaction. The IFA could be used in conjunction with any data base management system, or with a new DBMS whose actual data model is the SDM. When used with a conventional DBMS, a transaction output by the IFA would be translated into the query language of the conventional DBMS. Alternatively, if the IFA is used as a component of the user interface to a DBMS whose data model is the SDM, the interaction formalism transaction would be directly executed by the DBMS.

The interaction formulation advisor is application environment independent, in the sense that it can be used for any SDM data base. No modifications of the IFA are necessary to transport it to a new application: it is only necessary to define an SDM schema for the new application environment.

7.1. The IFA Interaction Formulation Methodology

The IFA is based on a specific model of the interaction formulation process; this model takes the form of a query formulation methodology. The IFA methodology is the basis for the design of the algorithm that underlies the SDM interaction formulation advisor. To describe this methodology, we proceed by first summarizing its major aspects. We then present several examples of user interactions with the IFA. Finally, we examine the principles underlying the IFA, consider its limitations, and discuss important extensions.

In interacting with the IFA, the user is focusing on the formulation of a query. He specifically performs the following actions, in the order indicated:

1. The user identifies the collection of entities in which he is interested (the *target class*). The user starts with some class in the data base schema, either one he knows is related to his query, or one that he chooses from a list provided by the IFA. He may then travel to other classes, eventually obtaining the target class. To go from the current *working class* to another *relevant class*, the IFA offers the user possible paths; a relevant class is one that the user feels is related to the target class, and which is closer to the target class than the working class is to the target class. The user is offered subclasses of the working class, superclasses of it, and so forth. The IFA also aids the user in

defining a relevant class if it is not already present in the schema.

2. The user specifies which aspects of the target class are of interest to him; he identifies the *target attributes*, which may include attributes of each member of the target class and/or attributes of the target class as a whole. The IFA assists the user in finding attributes that are of interest: the user first focuses on member atributes, and then considers class-determined and class attributes. The IFA uses the attribute inheritance rules of the SDM to provide the user with an "intelligent" list of possibilities for identifying the target attributes.

Derived information in an SDM schema has a very important use in the IFA: it allows the IFA to take advantage of the typical commonality among user interactions to simplify the user's job in formulating a new query. Many new uses of a data base are related to previous ones, and an SDM schema can evolve to match the needs of its users: the kinds of derived information that are frequently needed can be incorporated into an SDM schema.

7.2. The IFA Prototype

A prototype interaction formulation advisor (IFAP) has been designed and implemented. This prototype is based on the query formulation methodology described in section 7.1. The prototype (IFAP) guides a user through the formulation methodology. A user interacts with the IFAP, answering the questions that it asks; the user is provided with help information whenever he requires it.

In the following section, a number of transcripts of user interactions with the IFAP are examined in detail. In parallel with these examples, we describe the specific features of the IFA prototype.

7.3. Using the Interaction Formulation Advisor Prototype

As a first example of a user's interaction with the IFAP, let us suppose that some TMAE user, say an official in the U.S. Coast Guard, wishes to determine the name of each oil tanker registered in Liberia, as well as the name of its current captain. Suppose that this user has not asked the question before, and that he has very little experience with the IFA and the TMAE data base (TMDB). Appendix I-a contains a transcript of a dialogue between the user and the IFAP; the user here is a simulated Coast Guard Official. (Note that the transcript is exact, and was produced on a standard computer terminal; a variable character width text font is used in the appendix, which makes the alignment appear rough.)

The following is a description of the dialogue shown in figure I-a:

1. First, the user is asked to choose a data base. As it happens, a single data base was available for the IFAP test shown in the appendix: the tanker monitoring data base (TMDB).

2. To determine how much help information should be automatically provided by the IFAP, the user is asked if he is experienced (question S1). Help information can be explicitly requested at any time by typing a question mark ("?") or "help" in response to

a question asked by the IFAP.

3. The basic strategy of the interaction formulation process is explained to the user (if he doesn't already know it). As stated above, the strategy is for the user to describe some collection of entities (the target class) and then to identify the aspects of those entities in which he is interested (the target attributes). All queries are phrased in this way.

4. If the target class is defined in the schema and the user knows its name, he can simply state that it is the target class (question S2). If the user were able to do this, the IFAP would proceed to assist him in identifying the target attributes.

5. The IFAP tries to get the user to type in the name of a relevant class, i.e., one which is related to the collection of things in which he is interested (questions S3 - S4).

6. Since the user has not identified a relevant class, he is offered the base classes and asked to choose one of them with which to start (S5). Only the base classes are offered in order to make the number of choices of starting class manageable. All other classes in the schema can be reached from the base classes via interclass connections. Since he is interested in ships, the user identifies SHIPS as the starting class.

7. The user is asked if SHIPS is the target class (S6). Since it is not, it is considered the working class, i.e., the class the user currently has "in hand". The IFAP now proceeds to attempt to guide the user from this working class to another class that is the target class or that is closer to the target class.

8. The IFAP now considers that possibility that the user is interested in a subclass of SHIPS, i.e., a class that contains some, but not all of the members of the working class.

The subclasses offered to the user include classes that are restrictions or subsets of SHIPS, as well as those that are defined via "extract common members" or "extract missing members" thereon. Only immediate subclasses are presented in statement S9; subclasses of subclasses of SHIPS will be offered later. The user selects OIL_TANKERS as the new working class, since he is specifically interested in ships that are oil tankers.

9. The user now identifies LIBERIAN_OIL_TANKERS, a subclass of OIL_TANKERS, as the target class (in statements S11 - S13).

10. Now that the user has identified the target class, he can proceed to identify the target attributes. First, he is asked if he is interested in attributes of each member of LIBERIAN_OIL_TANKERS (S14). Since he is, the user is presented a list of all of the member attributes of LIBERIAN_OIL_TANKERS; the attribute inheritance rules of the SDM are used to compute this list. In this case, the user chooses attribute Name (S15).

11. The user states that he is also interested in the attribute Captain of LIBERIAN_OIL_TANKERS (S16, S17). This attribute does not have values that are strings or numbers, i.e., it is not a subclass of STRINGS or NUMBERS; therefore it cannot be directly printed. Consequently, information in the SDM schema is used to instead offer the user attributes of values of Captain, i.e., of members of OFFICERS, and to allow him to select one or more of these (S18 - S21). Here, the user selects Name of Captain.

12. After verifying that he is interested in the member attributes Name and Name of Captain (of class LIBERIAN_OIL_TANKERS) (S23), the system turns to attributes that are associated with the target class as a unit, rather than with each member of it. Specifically, the IFAP offers the user class attributes and class-determined attributes; again, the relevant attributes are determined using the SDM attribute inheritance rules. In this case, the user is not interested in any such attributes (S24, S25).

13. The transaction that the user has formulated, expressed in the interaction formalism, is now displayed. In this case it is a very simple one, since the user has decided to examine attributes of a class defined in the schema. The user is then asked if he wants to state another query, or if he is done with his session (S26, S27).

The query represented in the above interaction is a relatively easy one for the user to express: the derived information it requires is present in the SDM schema. In particular, the class LIBERIAN_OIL_TANKERS is already defined, as are the attributes Name, and Name of Captain. This is an example of the use of derived information to simplify the most common types of interactions.

Now let us examine how the IFAP handles a more complex query. Suppose that the user wishes to determine the hull number and last inspection date of each merchant Panamanian oil tanker. (We assume here that this is the same user as before). Appendix 1-b contains a transcript of an IFAP interaction for this query. We note here the significant aspects of the dialogue:

1. The user starts with the class LIBERIAN_OIL_TANKERS, which he remembers

from the previous interaction (S3, S4). Note that the system automatically adds the appropriate "_" characters to a user-supplied class name and appends an "S" to the end if necessary (S4). The user decides that he is interested in a superclass of LIBERIAN_OIL_TANKERS (S7), viz., OIL_TANKERS (S8).

2. The user now states that a subclass of OIL_TANKERS is relevant (S10); however, an appropriate subclass is not defined in the schema (S11). So, the user will be asked to define this relevant subclass, with the help of the IFAP (S12 - S14).

In addition to subclasses, IFAP user can define other kinds of classes as well. Corresponding to the SDM interclass connections, a class can be defined in the IFAP as:

a. a subclass of the working class,

b. a superclass of the working class,

c. a class that is a relative complement of the working class (with respect to some other class in the schema),

d. a class whose members are abstractions of the members of the working class.

3. Having decided to define a subclass, there are several ways in which the user can proceed: he can define the relevant subclass by any the following methods:

a. the new class is the intersection of the working class with some other class that is defined in the schema (using "extract common members"),

b. the new class is defined via "extract missing members" applied to the working class and some other class in the schema (the new class contains the members of the working class that are not in this other class),

c. the new class is defined by stating a predicate that a member of the working class must satisfy for it to be a member of the new class.

These three strategies are offered to the user in the order indicated here. If a strategy does not make sense, it is not offered to the user (as discussed in point 4 immediately below). In the example, the user succeeds in applying the first strategy (S15, S16); he decides to intersect the working class (OIL_TANKERS) with the class MERCHANT_SHIPS.

4. The semantic information provided by the SDM is used to offer the user only meaningful choices for use with "extract common members" and "extract missing members". The rules used to compute the meaningful options are identical to those specified for the applicability of the multiset operator interclass connections (see chapter 3). Namely, only classes that have a common underlying class with the working class are offered as options; SHIPS is this underlying class in the example interaction. Moreover, a class is not offered as an option if when used with "extract common members" or "extract missing members" it would yield a class that is the same as one the user has already rejected (e.g., in the example, LIBERIAN_OIL_TANKERS is eliminated for this reason).

5. The user is not yet satisfied with the working class, and he tries to obtain a further subclass of it (S17 - S20). This time, he is led to the third subclass definition strategy (S21- S25).

6. The user is now guided through the process of stating a predicate on the value of

some attribute of the members of the working class, i.e., of MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS]. First, the user selects the relevant attribute: Country_of_registry (S26). If he wished to state a complex predicate, e.g., involving multiple attributes, he would do so in steps by defining a sequence of subclasses. There are three main types of predicates that the user can state on the value of Country_of_registry:

a. that it must be equal to, not equal to, greater than, or similarly compared with some value the user can type in,

b. that it must be present (or not present) in some other class,

c. that it must be related in some specified way to some other attribute of members of the working class (e.g., equal to it).

7. Values of Country_of_registry are in the class COUNTRIES, and are neither strings nor numbers. Thus, the user cannot type in a value to which this attribute is to be compared. Rather, the user can specify a country by supplying a value for some attribute of it. In this case, the user selects attribute Name of COUNTRIES (S27, S28), and specifies that its value must be equal to "Panama" (S29 - S32). Note that the user could have selected the relevant country according to other criteria, e.g., by which ships are registered there. The flexibility exhibited here is a consequence of allowing SDM entities to stand for themselves rather than selecting some single identifier to stand for them as the value of an attribute.

8. The user has now identified the target class (S34), and goes on to identify the target

attributes: Hull_number and Date_last_examined (S35 - S41), in much the same way as in the preceding example.

9. The transaction that the user has defined contains two class definitions (one is an "intermediate"), and a specification of which of the member attributes of one of these classes are to be printed. (If information from several classes is to be printed, suitable derived attributes are used.)

As a third and final example of the IFAP, let us now suppose that the user wishes to do the following: for each inspection of a ship banned from U.S. coastal waters, find the name of the inspected ship and the type(s) of cargo it can carry. In this interaction, we assume that the user does not have a good idea of exactly what question he wants to ask; rather, as indicated in the transcript in appendix 1-c, the user is "browsing" through the schema, exploring its contents. We comment here on the most significant aspects of the dialogue:

1. The user starts with the class MERCHANT_SHIPS (S2 - S4). He decides that it is not the target class. He also rejects several possibilities (S5 - S10):

a. a subclass of MERCHANT_SHIPS,

b. a superclass of MERCHANT_SHIPS,

c. a class that consists of the members of some other data base class that are not in MERCHANT_SHIPS,

d. a class that contains abstractions of the members of MERCHANT_SHIPS,

e. a class that contains abstractions, one of which has its instances in class

MERCHANT_SHIPS (namely, the member of SHIP_TYPES that represents the "merchant ship" type).

2. Having rejected all of the possibilities offered by the IFAP, the user seems to be having some trouble in getting to the target class. The IFAP warns him that this is the case, and asks if the questions about MERCHANT_SHIPS should be repeated (S11); this gives the user the opportunity to change his mind and try one of suggested paths to another class, now that he is conscious of the alternatives. In the example interaction, the user decides that the questions should repeat, and chooses SHIPS, a superclass of the working class (S12 - S16).

3. The user now considers subclasses, abstraction classes, and aggregate classes of SHIPS, and rejects all of these (S17- S20). The option of going from SHIPS to a superclass of it is not offered to the user, since it is not possible to have a superclass of a base class (here, SHIPS); when an SDM schema is designed, base classes are defined so as to include all of the most general types of entities (see chapter 4).

4. The user is not making much progress towards a target class. Consequently, the IFAP suggests that he may be indirectly interested in SHIPS (S21- S22). The IFAP uses SDM schema information to trace all paths from SHIPS to attributes whose values are in class SHIPS or subclasses of SHIPS. In S23, the user is provided with a list of all such attributes. Since he is interested in inspections, the user selects attribute Tanker of INSPECTIONS. INSPECTIONS is now the new working class.

5. The user states that he is interested in defining a new subclass of INSPECTIONS by

stating a predicate on one of its attributes, viz., Tanker (S24 - S31). The user states that he cannot type in a value that some attribute of Tanker is to be compared to (S32), but that he wishes to constrain the value of Tanker to be present in some class (S33).

6. The goal now is for the user to identify the class that must contain the value of Tanker. To identify this *secondary target class*, a procedure analogous to the one used to find the (primary) target class is followed. In general, a secondary target class is identified in the same way as the (primary) target class; the IFA target class identification procedure is recursively applied. In the example interaction, the user selects BANNED_SHIPS (S34 - S41) as the secondary target class. Of course, the IFAP checks to make sure that it is meaningful and possible for a value of attribute Tanker of INSPECTIONS to be present in the class BANNED_SHIPS.

7. The IFAP now returns to consideration of the (primary) target class. The user has identified the target class as the class containing all inspections of banned ships ([INSPECTIONS]_WHERE_[TANKER_IS_IN_[BANNED_SHIPS]]) (S42 - S44).

8. The target attributes of members of the target class are identified as Name of Tanker and Cargo_types of Tanker (S45 - S53). Here, the user has selected two attributes of the value of attribute Tanker. The first is single-valued, while the second is multi-valued; the modifier "(all values of)" in S53 appears to indicate to the user that he will see all values of Cargo_types for the tanker. (This example illustrates the use of a mapping; for a complete discussion of mappings in the SDM, see chapter 3.)

Appendix I-d contains part of a dialogue that is a slight variation of that in appendix l-c. Here, the user is interested in formulating a transaction similar to the one specified in appendix 1-c, except that it deals with all inspections of tankers involved in oil spills, instead of all inspections of tankers that are banned. Here, the user has responded differently to S39 than he did in the dialogue in appendix 1-c. He has decided that the secondary target class is the subclass of SHIPS that consists of precisely those ships that are a value of attribute Involved_ship of OIL_SPILLS (for some member of OIL_SPILLS) (S44 -S48). This illustrates another important way of defining a subclass in the IFAP: the new class contains all of the members of the working class that are the value of some attribute of some other class in the schema. In the example interaction, the options offerred the user in S45 are those attributes in the data base whose values are in class SHIPS or a subset thereof.

Finally, appendix 1-e shows another variation of the dialogue in appendix 1-c. Here, the user has decided to repeat the query for all inspections of tankers whose captain was commissioned before his commander (presumably the user's measure of a mediocre captain). Here, the user is stating a predicate on the value of Captain of Tanker (a mapping) (S35 - S36), rather than a predicate on Tanker itself. Further, he is requiring the value of Captain to be in a class he will define (S37). He defines this class as the subclass of OFFICERS whose value of attribute Date_commissioned is less than or equal to the value of attribute Date_commissioned of the Commander of the officer (S47 - S62).

7.3.1. Observations on the IFAP Methodology

From the above examples, it is possible to make several important observations concerning the operation of the IFAP:

1. The IFAP is an *interactive* user interface tool. Questions are asked of the user, and he is required to provide only very simple types of responses: "yes" or "no", an option number, or a simple typed item (class or attribute name, or value). To conduct a dialogue with the user, the IFAP calls on a library of question templates, which are tailored to a given session.

2. All queries are formulated in a *stepwise* fashion: at every point in the interaction formulation process, the user focuses on a relatively simple task, e.g., finding a subclass of the working class. In identifying the target class, the user always works with a single working class. He can follow logical links to other classes from this working class, eventually homing in on the target class.

3. The IFAP *restricts the freedom* of the user, and guides him in formulating an interaction. Guidance is especially useful when the user is not sure exactly what question he wishes to ask. It is also advantageous when the user is involved in the formulation of a relatively complex definition, e.g., when he is defining a new class. In defining a class not in the schema, the user follows a specific disciplined procedure, which encourages him to try the easiest options first. The user is also limited in the number of distinct ways he can define the same class. For example, a subclass of the working class can be defined via "merge members" or "extract common members"

applied to a class defined in the schema, but cannot be applied to a user-defined class; subclasses of the latter kind are instead defined by stating a constraint on the members of the working class (e.g., a predicate on the values of its member attributes). These techniques provide a good deal of control of the degrees of freedom that are available for defining a new class; we believe that this is a major improvement over existing data base query systems.

4. *Derived information* in an SDM schema has an important impact on the IFAP dialogue. If relevant derived information is present in an SDM schema, the query formulation task is significantly simpler. The idea is that frequently used derived information should be integrated into the SDM schema, so that it is readily available. When new derived information is needed, the user is called upon to do more work, and to define it himself; his task then is somewhat more difficult, but the IFAP provides guidance to assist him.

As specified in chapter 5, new derived information defined within a user's interaction only exists during that IFAP interaction. The determination of when and if this information should migrate into the schema is made by the data base administrator.

7.4. The IFAP Implementation

The interaction formulation advisor prototype, as implemented, performs two functions:

1. It accepts as input an SDM schema, which it parses and checks for legality and

consistency. A schema is defined for each application environment and data base that is to be used with the IFAP; these schemas are placed in files that are scanned by the IFAP. Many kinds of checking are performed by the IFAP, including making sure that all referenced class and attribute names are defined, checking that all class and attribute derivations are legal, etc. An internal form for the SDM schema is produced, which is used to support the second function.

2. The IFAP conducts an interactive dialogue with the user, guiding him through the schema and helping him formulate a transaction.

As specified above, the IFAP is application environment independent. No "tuning" is required to use the IFAP for a new application: only a new SDM schema need be defined. This means that the IFAP is very transportable, much more so, say, than the existing natural language query systems. In fact, the IFAP has been successfully demonstrated for three applications: tanker monitoring, aircraft maintenance records and scheduling, and bibliographic information for industrial energy conservation.

The prototype interaction formulation advisor was implemented in CLU, a new programming language developed at MIT [Liskov 1977, Liskov 1978]. The prototype runs on a DEC PDP-10 under the MIT "home grown" ITS operating system, and also runs on the DECSYSTEM-20 under the TOPS-20 operating system. The IFAP implementation is operating system independent, since it relies exclusively on the facilities provided by the CLU compiler and the CLU support system (run time environment package).

Appendix 2 contains a very short sample of CLU code from the IFAP implementation. The excerpt shown there is a module that is invoked early in an IFAP dialogue when the user cannot name the target class and does not know a relevant class. This procedure takes the following arguments: the console input and output files, the SDM schema internal form, a flag indicating wh‹ her or not the user is experienced, and the number of the question last asked. The procedure returns a class and a new question number. "Bool", "int", etc. are built-in CLU types, while "schema" and "class" are user-defined types (specific to the IFAP implementation). The module iterates over the base classes in the schema, presents them as options to the user, and allows him to select one of them as the starting class.

In all, there are 6411 lines of CLU source code in the IFAP implementation; there are 6780 lines when comments are included. Roughly half of this code is concerned with parsing and checking an SDM schema, and with producing appropriate internal structures. The other half is concerned with conducting the dialogue itself; included here is code to determine which questions should be asked. The code occupies about 51K of memory when it is compiled and linked. When the CLU run time support system is added, the total memory occupied by IFAP is 107K.

The example SDM schema used in this chapter (the TMDB schema) contains 276 lines, and defines 34 classes. To parse, check, and load this schema takes about 24 seconds of CPU time on a DEC KA-10, and about 9 seconds on a DEC 2050. A typical dialogue, such as the one in appendix I-b, takes about the same amount of run time.

The CLU language and CLU system were helpful tools in getting the IFAP working in a relatively short amount of time. CLU is a structured language that directly supports data abstractions (user-defined data types) and control abstractions (user-defined iterators), as well as the more standard procedural abstractions. The strong type checking principle upon which the compiler is based makes initial debugging easy, and the library system (not complete) simplifies the management of the inevitable changes in design during implementation.

7.5. Limitations and Directions for Further Work

We believe that the IFA prototype and the model of the query formulation process that underlies it represent an initial version of an effective user interface tool for naive users. However, the prototype is incomplete in a number of ways. In this section, we comment on what we believe to be the major limitations of the IFAP, and describe directions for further work.

7.5.1. Experience with the IFAP

With our initial experience with the IFA prototype, it is possible to comment on the most important modifications and extensions that should be made in future versions of the system. Further experimentation with real users is necessary to resolve some of the specific issues, and would be useful in any event to evaluate the effectiveness of the IFAP. Here, we note some minor improvements that should be made to the IFAP, and discuss longer-term *extensions in the following section.*

1. The names of user-defined classes in the IFAP are generated by the system. These names become somewhat unwieldy when a user performs complex sorts of class definitions. This problem could be alleviated by replacing a class name with a paraphrase of its meaning, or by allowing the user to optionally give a name for any new class he defines.

2. The user should be allowed to select more than one option in a single response, when that makes sense. For example, if he wishes to select two target member attributes, he should be allowed to choose both of them at the same time. The IFAP should also allow the user to type the text of the option, rather than its number, if the user so prefers. For example, instead of forcing the user to type the number of a class name in a list of options, he should also be allowed to type the name of the class itself.

3. There are several places in the IFAP algorithm where more control is needed to allow a user to go back and start again at some point in the interaction. This would allow him to more easily follow a different path from some given point in an interaction, or to correct a mistake that he made. He can do this now, but only in limited ways.

4. It would be useful to allow a user to save a transaction he has formulated, so that he can use it again later. It would also be helpful for a user to be able to reload classes that he defined in a previous transaction if he finds them useful in the current interaction.

5. The specific language used in the dialogue should be improved to be more user-friendly. For example, it may be useful to use terms like "information on ships" rather

"attributes of ships", etc. In general, this is a nontrivial problem, and necessitates psychological experimentation with users.

7.5.2. Extensions

In addition to these specific revisions, there are a number of areas for extension of the IFA. Some of these extensions will be implemented in future versions of the system. We list these extensions here, roughly in decreasing order of importance:

1. The IFA prototype does not allow the user to define new attributes; attributes that are referenced by the user must be defined in the SDM schema. One way to enhance the power of the IFA vis-a-vis attributes is to automatically compute inversions and other types of derived attributes, and list them in attribute option lists. More ambitiously, the user can be allowed to define any new attribute. This would be implemented by offering him the various types of attribute derivation specifications, and guiding him through the process of defining the new attribute. The methodology used for this would be analogous to that used to allow users to define new classes.

2. A more substantial facility for aiding the user in assessing his current position in the overall interaction formulation process would be helpful to a naive user. This facility should allow the user to:

   a. determine "where he is" in the interaction formulation process, e.g., by providing him a description of the IF transaction he has constructed thus far,

   b. request additional information on what his subsequent options will be if he chooses a particular option now,

c. obtain additional information about a potentially relevant class or attribute.

3. The IFA algorithm should be extended to accomodate several additional types of query structures:

a. It should be possible for a user to specify predicates involving logical "and" and "or" in a direct fashion. Currently, he must define a sequence of subclasses to specify this kind of restriction.

b. The interattribute comparison facility for subclass definition should be extended to accomodate set comparisons.

c. A more flexible facility is needed to handle set comparisons involving mappings (e.g., for INSPECTIONS, "Tanker.Cargo_types contains 'oil'").

d. The handling of multiset operators should be extended to allow a user to define a new class to be used in conjunction with "merge members", "extract common members", or "extract missing members" and the working class. Currently, only classes defined in the schema can be used in combination with the multiset operators. This extension must be carefully integrated with the overall IFA strategy, in order to avoid diverting a user to trying to define a complex intersection, union, or difference, when another way of defining a new class would be easier (e.g., a restriction).

4. More of the semantic information captured in an SDM schema should be used in the IFA. For exampie, the IFA should be "smarter" about attribute derivations, restriction

predicates, etc.; currently, only limited analysis of restriction predicates is performed. For example, the IFA could determine that the class LIBERIAN_OIL_TANKERS is a restriction of SHIPS (with the predicate "where Cargo_types contains 'oil' and Country.Name = 'Liberia'", as well as a restriction of OIL_TANKERS (with the predicate "where Country.Name = 'Liberia'").

5. An extended model of the query formulation process should be developed, in order that the IFA (or a descendant of it) easily accommodate a wider variety of query types.

6. The IFA should be interfaced with external transaction facilities, e.g., so that it can call on external programs when necessary. For example, in a data base used to monitor ship positions, it might be useful to have a transaction that given a ship, returns the ship closest to it. (Here, "closest" means having the shortest sailing time.) Such a transaction must of course have a knowledge of possible sailing courses, locations of land masses, and so forth.

7. User profiles should be used to record the progress of a user in learning about the IFA and the data base, and to tailor the dialogue to his needs.

8. The IFA should be extended to allow the user to define new transactions that update the data base.

9. Variations of, and alternative approaches to, the IFA should be investigated. For example, it is important to study the effectiveness of the IFA in accomodating a data base browser who doesn't really want to state a query or update request, but is only interested in what information is in a data base. We note in this regard that the IFA

schema parser, checker, and internal schema form generator can be used to support tools other than the dialogue component of the IFAP.

8. SUMMARY, CONCLUSIONS, AND DIRECTIONS FOR FURTHER WORK

The SDM has been designed to enable a computerized data base to directly model an application environment, by supporting data base structures that closely correspond to the natural constructs of that application environment. Our work has focused on the problems of increasing the understandability of a data base and on enabling it to be more accessible to users.

The SDM is based on a rich but limited set of semantic structure types; this vocabulary of structures allows a designer to construct a data base schema (description) containing primitives appropriate to the application domain (e.g., concrete objects, events, aggregates, abstractions, etc.). Important types of semantic integrity constraints are directly integrated into the basic structure of an SDM data base. In an important sense, an SDM schema is self-documenting. The definition of an SDM data base is couched in terms that should enable a user to understand its structure and to interact with it in a natural and convenient fashion.

We see redundancy as a natural aspect of data base applications, and the SDM provides a direct approach to the description of derived data. Derived information is as prominent as primitive data in an SDM schema. The specification of derived information is essentially equivalent to the formulation of queries on the data base. Thus, derived information in an SDM schema can be used to capture frequently retrieved data, and thereby simplify the users' most frequent interactions with the data base. The design of an SDM schema should provide users with the structures most likely to be useful in the formulation of interactions with the data base.

In this thesis, we have described the SDM in detail, and presented an SDM data definition language (schema definition language). Modelling with the SDM has been examined, comparisons made with conventional data models, and a design methodology for SDM data bases presented. Operations on an SDM data base have been specified, and an interaction formalism for the specification of user-defined transactions has been designed. As we have seen, the SDM has several important applications, including use as a formal specification mechanism, as a support for data base design tools, and as a basis for powerful user interface facilities.

In the preceding chapters, we have examined SDM user interface tools in some detail. First, conventional user interfaces to data base management systems were analyzed, and it was observed that the conventional approaches have a number of significant problems. We noted that a typical data base user community includes a wide variety of user types, with varying needs and abilities. The ways in which the SDM can support a structured user interface for these user communities were discussed; a spectrum of data base interaction strategies is required in a complete user interface, ranging from supporting the invocation of an appropriate transaction to the definition of new data base structures based on primitive SDM operations (modification of the schema). This structured user interface strategy is used to address the problem of handling both highly repetitive and one-time interactions with a data base.

The support of users who are naive of the content and structure of a data base was studied in detail. An interaction formulation advisor for naive users based on the SDM was designed and a prototype implementation developed. This user help facility provides interactive support for naive nonprogrammers, guiding them through the process of stating a query on a data base.

The SDM is seen as a vehicle for the specification of data semantics; the SDM can be used as the data model for a new SDM DBMS, can be used to support user interface tools like the IFA, and can aid the data base designer in the process of designing conventional data bases. It is thus not a competitor with, but a complement to, conventional data base models and systems.

8.1. Directions for Further Work and Extensions

Throughout this thesis, several areas for extending the work described herein have been proposed. We summarize here the areas of extension which are currently underway or anticipated:

1. Although a good deal of effort has gone into designing and revising the SDM, it will no doubt continue to evolve. As further experience is gained with new applications (e.g., a more detailed study of the SDM as a data base design aid), less useful features may be dropped and new features added. Our experience with the interaction formulation advisor prototype has shown that the specific features desirable in the SDM depend on the uses of the SDM.

2. The interaction formulation advisor and prototype implementation will be extended,

as described in chapter 7.

3. Further work on SDM data base design is in progress. This includes a more detailed analysis of the problem of SDM schema design as well as a study of the problem of mapping SDM data structures and operations into those of a conventional data model. The design and possible implementation of a data base design aid based on the SDM is anticipated.

There are several other areas in which our work can be extended; the most important of these are as follows:

1. The study of a "semantic data dictionary" based on the SDM should be undertaken (see chapter 7 for details).

2. The integration of SDM user interface tools with decision support systems, management information systems, and office automation tools should be studied.

3. Interfacing the SDM interaction formalism with general purpose programming languages and related matters should be considered.

4. It is important to consider how natural language processing techniques can be used in conjunction with the interaction formulation advisor. The utility of the SDM in supporting various types of generalized knowledge representation should also be assessed.

5. The use of the SDM for expressing a wide-variety of types of constraints should be considered, e.g., including semantic integrity constraints, protection restrictions, etc.

6. The SDM provides some interesting possibilities for impact on improved query

processing capabilities, including:

a. using the semantic information captured in the SDM for query optimization (the optimization of strategies and procedures to produce answers to queries),

b. using the SDM in attacking the problem of query approximation (the generation of low cost, approximate answers to queries).

REFERENCES AND DESCRIPTIVE BIBLIOGRAPHY

Key Phrases

Each document cited has been assigned one or more key phrases to describe its contents (as it relates to the theme of this bibliography). The abbreviation for the key phrase (the first letter of each word in the phrase) is listed in angle brackets ("<>") with each citation. The following list of key phrases and corresponding abbreviations is used:

A - applications
DM - data models
DI - deductive inference
FD - functional dependencies
GI - general issues
HDM - hierarchical data model
I - implementations
IR - information redundancy
LD - logical design (modelling)
MUV - multiple user views
NLI - natural language interaction
NDM - network data model
NI - nonprocedural interaction (very high level languages)
PI - procedural interaction
RDM - relational data model
S - semantics (of data)
SI - semantic integrity
UC - user characteristics
UR - user requirements

[Abbey 1976]
Abbey, S. and S. Lipka, "Implementation Independent Queries and Retrievals", *Proceedings of Fifth Texas Conference on Computing Systems*, Austin TX, 18-19 October 1976.
<NI, PI>

[Abrial 1974]
Abrial, J. R., "Data Semantics", *Data Base Management* (editors Klimbie, J. W. and K. L. Koffeman), North Holland, 1974.
<DM, S>

[Adiba 1976]
Adiba, M., C. Delobel, and M. Leonard, "A Unified Approach for Modelling in Logical Data Base Design", *Modelling in Data Base Management Systems* (editor Nijssen, G. M.), North Holland, 1976.
<LD>

[Allman 1976]
Allman, E., M. Stonebraker, and G. Held, "Embedding a Relational Data Sublanguage in a General Purpose Programming Language", *Proceedings of ACM SIGPLAN/SIGMOD Conference on Data: Abstraction, Definition, and Structure*, Salt Lake City UT, 22-24 March 1976.
<PI, RDM>

[Ames 1977]
Ames, S. R. Jr., "User Interface Multilevel Security Issues in a Transaction-Oriented Data Base Management System", *Proceedings of Conference on Computer Security and Integrity - Trends and Applications*, Gaithersburg MD, 19 May 1977.
<NI>

[Anderson 1978]
Anderson, N. D. and W. A. Burkhard, "MINISEQUEL - Relational Data Management System", *Proceedings of International Conference on Data Bases: Improving Usability and Responsiveness*, Haifa, Israel, 2-4 August 1978.
<I, NI, RDM>

[ANSI/X3/SPARC 1975]
ANSI/X3/SPARC (Standards Planning and Requiiements Committee), "Interim Report from the Study Group on Data Base Management Systems", *FDT* (Bulletin of ACM SIGMOD), Volume 7, Number 2, 1975.
<DM, LD, MUV, UR>

[Appleton 1977]
Appleton, D. S., "What Data Base Isn't", *Datamation*, January 1977.
<GI>

[Armstrong 1974]
Armstrong, W. W., "Dependency Structures of Data Base Relationships", *Information Processing '74*, North Holland, 1974.
<DM, FD>

[Arora 1978]
Arora, A. K. and R. Carlson, "The Information Preserving Properties of Relational Database Transformations", *Proceedings of International Conference on Very Large Data Bases*, West Berlin, West Germany, 13-15 September 1978.
<FD, RDM>

[Ash 1968]
Ash, W. L. and E. H. Sibley, "TRAMP: An Interpretive Processor with Deductive Capabilities", *Proceedings of ACM National Conference*, Las Vegas NV, 27-29 August 1968.
<DI>

[Ashenhurst 1975]
Ashenhurst, R. L. and R. H. Vanderohe, "A Hierarchical Network", *Datamation*, Volume 21, Number 2, Pages 40-44, February 1975.
<DM, HDM, NDM>

[Astrahan 1975a]
Astrahan, M. M. and D. D. Chamberlin, "Implementation of a Structured English Query Language", *Communications of the ACM*, Volume 18, Pages 580-588, 1975.
<I, NI, RDM>

[Astrahan 1975b]
Astrahan, M. M. and R. A. Lorie, "SEQUEL-XRM: A Relational System", *Proceedings of ACM Pacific Conference*, San Francisco CA, 17-18 April 1975.
<I, RDM>

[Astrahan 1976]
Astrahan, M. M., M. W. Blasgen, D. D. Chamberlin, K. P. Eswaran, J. N. Gray, P. P. Griffiths, W. F. King, R. A. Lorie, P. R. McJones, J. M. Mehl, G. R. Putzolu, I. L. Traiger, B. W. Wade, and V. Watson, "System R: A Relational Approach to Data Base Management", *ACM Transactions on Database Systems*, Volume 1, Number 2, June 1976.
<I, MUV, NI, PI, RDM, SI>

[Atkinson 1978]
Atkinson, M. P., "Programming Languages and Databases", *Proceedings of International Conference on Very Large Data Bases*, West Berlin, West Germany, 13-15 September 1978.
<PI>

[Aurdal 1978]
Aurdal, E. and A. Solvberg, "A Multilevel Procedure for Design of File Organizations", *Proceedings of National Computer Conference*, Dallas TX, 13-16 June 1977.
<DM, LD>

[Bachman 1969]
Bachman, C. W., "Data Structure Diagrams", *Data Base* (Bulletin of ACM SIGBDP), Volume 1, Number 2, Pages 4-10, 1969.
<LD, NDM>

[Bachman 1973]
Bachman, C. W., "The Programmer as Navigator", *Communications of the ACM*, Volume 16, Number 11, November 1973.
<NDM, PI>

[Bachman 1975]
Bachman, C. W., "Trends in Data Base Management", *Proceedings of National Computer Conference*, Anaheim CA, 19-22 May 1975.
<GI>

[Bachman 1977a]
Bachman, C. W., "Why Restrict the Modelling Capability of CODASYL Data Structure Sets?", *Proceedings of National Computer Conference*, Dallax TX, 13-16 June 1977.
<LD, NDM>

[Bachman 1977b]
Bachman, C. W., "The Role Concept in Database Models", *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
<DM, NDM, S>

[Badal 1977]
Badal, D. Z., *On Semantic Integrity in Centralized and Distributed Database Systems*, Technical Report, Computer Science Department, University of California, Los Angeles CA, 1977.
<SI>

[Bandurski 1975]
Bandurski, A. E. and D. K. Jefferson, "Data Description for Computer-Aided Design", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, San Jose CA, 14-16 May 1975.
<A, LD>

[Beck 1976]
Beck, L. L., "An Approach to the Creation of Structured Data Processing Systems", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Washington D.C., 2-4 June 1976.
<A>

[Beeri 1977]
Beeri, C., "A Complete Axiomatization for Functional and Multivalued Dependencies in Database Relations", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Toronto, Canada, 3-5 August 1977.
<FD, S>

[Bell 1968]
Bell, A. and M. R. Quillian, "Capturing Concepts in a Semantic Net", *Proceedings of Symposium on Associative Information Techniques*, Warren MI, 30 September - 1 October 1968.
<NLI, S>

[Benci 1976]
Benci, E., F. Bodart, H. Bogaert, and A. Cabanes, "Concepts for the Design of a Conceptual Schema", *Modelling in Data Base Management Systems* (editor Nijssen, G. M.), North Holland, 1976.
<DM, LD>

[Berild 1977]
Berild, S. and S. Nachmens, "CS4: A Tool for Database Design by Infological Simulation", *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
<LD>

[Berman 1976]
Berman, W. J., *Automatic Generation of Database Definitions for an Information Retrieval System*, Ph.D. Thesis, Stanford University, Palo Alto CA, 1976.
<LD>

[Bernstein 1975a]
Bernstein, P. A., J. R. Swenson, and D. C. Tsichritzis, "A Unified Approach to Functional Dependencies and Relations", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, San Jose CA, 14-16 May 1975.
<FD, LD, RDM>

[Bernstein 1975b]
Bernstein, P. A., *Normalization and Functional Dependencies in the Relational Model*, Technical Report CSRG-60, Computer Systems Research Group, University of Toronto, Toronto, Canada, October 1975.
<FD, LD, RDM>

[Bernstein 1975c]
Bernstein, P. A., *Normalization and Functional Dependencies in the Relational Data Base Model*, Ph.D. Thesis, Department of Computer Science, University of Toronto, Toronto, Canada, 1975.
<FD, LD, RDM>

[Bernstein 1976]
Bernstein, P. A., "Synthesizing Third Normal Form Relations from Functional Dependencies", *ACM Transactions on Data Base Systems*, Volume 1, Number 4, Pages 277-298, December 1976.
<FD, RDM>

[Biller 1974]
Biller, H. and E. Neuhold, "Formal View on Schema - Subschema Correspondence", *Information Processing '74*, North Holland, 1974.
<DM, MUV>

[Biller 1978]
Biller, H. and E. J. Neuhold, "Semantics of Data Bases: The Semantics of Data Models", *Information Systems*, Volume 3, Number 1, Pages 11-30, 1978.
<DM, S>

[Bjorner 1973]
Bjorner, D., E. F. Codd, K. L. Deckert, and I. L. Traiger, *The Gamma-0 N-ary Relational Data Base Interface Specifications of Objects and Operations*, IBM Research Report RJ1200, San Jose CA, 11 April 1973.
<I, RDM>

[Blasgen 1977]
Blasgen, M. W. and K. P. Eswaran, "Storage and Access in Relational Data Bases", *IBM*

Systems Journal, Volume 16, Number 4, Pages 363-377, 1977.
<RDM>

[Bledsoe 1974]
Bledsoe, W. W. and P. Bruell, "A Man - Machine Theorem-Proving System", *Artificial Intelligence*, Volume 5, Number 1, Pages 51-72, 1974.
<DI>

[Bobrow 1977a]
Bobrow, D. G., T. Winograd, and KRL Research Group, "Experience with KRL-0: One Cycle of a Knowledge Representation Language", *Proceedings of International Joint Conference on Artificial Intelligence*, Cambridge MA, 22-25 August 1977.
<S>

[Bobrow 1977b]
Bobrow, D. G. and T. Winograd, "An Overview of KRL, a Knowledge Representation Language", *Cognitive Science*, Volume 1, Number 1, January 1977.
<DI, S>

[Borgida 1975]
Borgida, A. T., *Topics in the Understanding of English Sentences by Computer*, Technical Report 78, Department of Computer Science, University of Toronto, Toronto, Canada, February 1975.
<NLI>

[Borkin 1978]
Borkin, S. A., *Data Model Equivalence" Proceedings of International Conference on Very Large Data Bases*, West Berlin, West Germany, 13-15 September 1978.
<DM HDM, NDM, RDM>

[Boyce 1973a]
Boyce, R. F. and D. D. Chamberlin, *Using a Structured English Query Language as a Data Definition Facility*, IBM Research Report RJ1318, San Jose CA, 10 December 1973.
<LD, NI, RDM>

[Boyce 1973b]
Boyce, R. F., D. D. Chamberlin, W. F. King III, and M. M. Hammer, "Specifying Queries as Relational Expressions: SQUARE", *Proceedings of ACM SIGPLAN-SIGIR Interface Meeting*, Gaithersburg MD, 4-6 November 1973.
<NI, RDM>

[Boyce 1974]
Boyce, R. F., D. D. Chamberlin, W. F. King III, and M. M. Hammer, "Specifying Queries as Relational Expressions: SQUARE", *Data Base Management* (editors Klimbie, J. W. and K. L. Koffeman), North Holland, 1974.
<NI, RDM>

[Boyce 1975]
Boyce, R. F., D. D. Chamberlin, W. F. King III, and M. M. Hammer, "Specifying Queries as Relational Expressions: The SQUARE Data Sublanguage", *Communications of the ACM*, Volume 18, Number 11, November 1975.
<NI, RDM>

[Bracchi 1972]
Bracchi, G. A., A. Fedeli, and P. Paolini, "A Language for a Relational Data Base Management System", *Sixth Annual Princeton Conference on Information Sciences and Systems*, Princeton NJ, 23-24 March 1972.
<NI, RDM>

[Bracchi 1974]
Bracchi, G., A. Fedeli, and P. Paolini, "A Multi-Level Relational Model for Data Base Management Systems", *Data Base Management* (editors Klimbie, J. W. and K. L. Koffeman), North Holland, 1974.
<I, RDM>

[Bracchi 1976]
Bracchi, G., P. Paolini, and G. Pelagatti, "Binary Logical Associations in Data Modelling", *Modelling in Data Base Management Systems* (editor Nijssen, G. M.), North Holland, 1976.
<LD, S>

[Brachman 1976]
Brachman, R. J., "What's in a Concept: Structural Foundations for Semantic Networks", *Proceedings of COLING Conference*, Ottawa, Canada, 1976.
<S>

[Bradley 1978]
Bradley, J., "Operations Databases", *Proceedings of International Conference on Very Large Data Bases*, West Berlin, West Germany, 13-15 September 1978.
<DM, S>

[Brodie 1975]
Brodie, M. C., S. S. Chan, B. Czarnik, E. Leong, S. A. Schuster, and D. C. Tsichritzis, *ZETA: A Prototype Relational Data Base Management System*, Technical Report CSRG-51,

Computer Systems Research Group, University of Toronto, Toronto, Canada, 1975.
<I, RDM>

[Brodie 1976]
Brodie, M. L. and D. Tsichritzis, *Data Base Constraints*, Technical Report, Department of Computer Science, University of Toronto, Toronto, Canada, 1976.
<RDM, SI>

[Brown 1975]
Brown, A. P. G., "Modelling a Real World System and Designing a Schema to Represent It", *Data Base Description* (editors Douque, B. C. M. and G. M. Nijssen), North Holland, 1975.
<LD>

[Bubenko 1976a]
Bubenko, J. A., S. Berild, E. Lindencrona-Ohlin, and S. Nachmens, "From Information Requirements to DBTG-Data Structures", *Proceedings of ACM SIGPLAN/SIGMOD Conference on Data: Abstraction, Definition, and Structure*, Salt Lake City UT, 22-24 March 1976.
<LD, NDM>

[Bubenko 1976b]
Bubenko, J. A., *The Temporal Dimension in Information Processing*, IBM Research Report RC6187, Yorktown Heights NY, 16 November 1976.
<LD, S>

[Bubenko 1977a]
Bubenko, J. A., *IAM: Inferential Abstract Modelling - An Approach to Design of Information Models for Large Shared Data Bases*, IBM Research Report RC6343, Yorktown Heights NY, 4 January 1977.
<LD>

[Bubenko 1977b]
Bubenko, J. A., "IAM: An Inferential Abstract Modelling Approach to Design of Conceptual Schema", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Toronto, Canada, 3-5 August 1977.
<DM, LD>

[Bubenko 1977c]
Bubenko, J. A., "Validity and Verification Aspects of Information Modelling", *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
<LD, S>

[Bukhari 1975]
Bukhari, S. A., *A Relation Algebra Model for Data Bases*, Technical Report CS-75-11, Computer Science Department, University of Waterloo, Waterloo, Canada, April 1975.
<DM, NI, RDM>

[Buneman 1977]
Buneman, O. P. and H. L. Morgan, "Implementing Alerting Techniques in Database Systems", *Proceedings of COMPSAC '77*, Chicago IL, 8-11 November 1977.
<SI>

[Burger 1977]
Burger, J. F., "Data Base Semantics in the EUFID System", *Proceedings of the Second Berkeley Workshop on Distributed Data Management and Computer Networks*, Berkeley CA, 25-27 May 1977.
<NI, S>

[Cadiou 1975]
Cadiou, J. M., "On Semantic Issues in the Relational Model of Data", *Proceedings of International Symposium on Mathematical Foundations of Computer Science*, Gdansk, Poland, September 1975.
<RDM, S>

[Canning 1972a]
Canning, R. G., "The Data Administrator Function", *EDP Analyzer*, Volume 10, Number 11, November 1972.
<A, GI>

[Canning 1972b]
Canning, R. G., "The Debate on Data Base Management", *EDP Analyzer*, Volume 10, Number 3, March 1972.
<GI, HDM, NDM, RDM>

[Canning 1974a]
Canning, R. G., "Problem Areas in Data Management", *EDP Analyzer*, Volume 12, Number 3, March 1974.
<GI, UR>

[Canning 1974b]
Canning, R. G., "What's Happening with CODASYL - Type DBMS?", *EDP Analyzer*, Volume 12, Number 10, October 1974.
<NDM>

[Carlson 1976]
Carlson, C. R., and R. S. Kaplan, "A Generalized Access Path Model and Its Application to a Relational Data Base System", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Washington D.C., 2-4 June 1976.
<DI, NI, RDM>

[CCA 1975]
Computer Corporation of America, *Datacomputer Version 1 User Manual*, Datacomputer Project Working Paper No. 11, Cambridge MA, 1 August 1975.
<HDM, I>

[Chamberlin 1974]
Chamberlin, D. D. and R. F. Boyce, "SEQUEL: A Structured English Query Language", *Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control*, Ann Arbor MI, 1-3 May 1974.
<NI, RDM>

[Chamberlin 1975]
Chamberlin, D. D., J. N. Gray, and I. L. Traiger, "Views, Authorization, and Locking in a Relational Data Base System", *Proceedings of National Computer Conference*, Anaheim CA, 19-22 May 1975.
<MUV, NI, RDM>

[Chamberlin 1976a]
Chamberlin, D. D., "Relational Data-Base Management Systems", *Computing Surveys*, Volume 8, Number 1, March 1976.
<RDM>

[Chamberlin 1976b]
Chamberlin, D. D., M. M. Astrahan, K. P. Eswaran, P. P. Griffiths, R. A. Lorie, J. W. Mehl, P. Reisner, and B. W. Wade, *SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control*, IBM Research Report RJ1798, San Jose CA, 23 June 1976.
<LD, MUV, NI, RDM, SI>

[Chamberlin 1976c]
Chamberlin, D. D., M. M. Astrahan, K. P. Eswaran, P. P. Griffiths, R. A. Lorie, J. W. Mehl, P. Reisner, and B. W. Wade, "SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control", *IBM Journal of Research and Development*, Volume 20, Number 6, Pages 560-575, Novenber 1976.
<LD, MUV, NI, RDM, SI>

[Chan 1974]
Chan, S. S., *QLS: A Query Language Generator System*, M.S. Thesis, Department of Computer Science, University of Toronto, Toronto, Canada, 1974.
<NI>

[Chan 1976]
Chan, A. Y., *Index Selection in a Self-Adaptive Relational Data Base Management System*, Technical Report TR-166, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge MA, September 1976.
<RDM>

[Chang 1973]
Chang, C. L. and R. C. T. Lee, *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, 1973.
<DI>

[Chang 1975]
Chang, C. L., *A Hyper-Relational Model of Data Bases*, IBM Research Report RJ1634, San Jose CA, 22 August 1975.
<LD, NI, RDM, S>

[Chang 1976]
Chang, C. L., "DEDUCE - A Deductive Query Language for Relational Data Bases", *Pattern Recognition and Artificial Intelligence* (editor Chen, C. H.), Academic Press, 1976.
<LD, NI, RDM, S>

[Chang 1978a]
Chang, C. L., *Finding Missing Joins for Incomplete Queries on Relational Data Bases*, IBM Research Report RJ2145, San Jose CA, January 1978.
<NI, RDM, S>

[Chang 1978b]
Chang, S. K. and J. S. Ke, "Database Skeleton and Its Application to Fuzzy Query Translation", *IEEE Transactions on Software Engineering*, Volume SE-4, Pages 31-44, January 1978.
<LD, NI, RDM, S>

[Chang 1978c]
Chang, S. K. and W. H. Cheng, "Database Skeleton and Its Application to Logical Database Synthesis", *IEEE Transactions on Software Engineering*, Volume SE-4, Pages 18-30, January 1978.
<LD, NI, RDM, S>

[Chen 1976]
Chen, P. P. S., "The Entity - Relationship Model: Toward a Unified View of Data", *ACM Transactions on Data Base Systems*, Volume 1, Number 1, Pages 9-36, March 1976.
<DM, LD, S>

[Chen 1977a]
Chen, P. P. S., "The Entity - Relationship Model: A Basis for the Enterprise View of Data", *Proceedings of National Computer Conference*, Dallax TX, 13-16 June 1977.
<A, DM, LD, S>

[Chen 1977b]
Chen, P. and B. Yao, "Performance and Design Aids for Data Base Systems", *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
<LD>

[Chen 1978a]
Chen, P., *The Entity-Relationship Approach to Logical Data Base Design*, Monograph Number 6, QED Information Sciences, Wellesley MA, 1978.
<DM, LD, S>

[Chen 1978b]
Chen, P. S., "Applications of the Entity-Relationship Model", *Proceedings of NYU Symposium on Database Design*, New York NY, 18-19 May 1978.
<DM, LD, S>

[Childs 1968a]
Childs, D. L., "Feasibility of a Set-Theoretical Data Structure - A General Structure Based on a Reconstituted Definition of a Relation", *Information Processing '68*, North Holland, 1968.
<RDM>

[Childs 1968b]
Childs, D. L., "Description of a Set-Theoretic Data Structure", *Proceedings of Fall Joint Computer Conference*, 1968.
<RDM>

[Childs 1977]
Childs, D. , "Extended Set Theory", *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
<LD, RDM>

[Clark 1976]
Clark, I. A., *STREMA: A Graphic Language for Relational Applications*, IBM Scientific Centre Report UKSC-0084, Peterlee, England, 1976.
<NI, RDM>

[Claybrook 1976]
Claybrook, B., "The Design of a Template Structure for a Generalized Data Structure Definition Facility", *Proceedings of Second International Conference on Software Engineering*, San Francisco CA, 13-15 October 1976.
<LD>

[Clemons 1977]
Clemons, E. K., *Design of a User Interface for a Relational Data Base*, Technical Report 76-09-08, Department of Decision Science, The Wharton School, University of Pennsylvania, Philadelphia PA, 1977.
<NI, PI, RDM>

[Clemons 1978]
Clemons, E. K., "An External Schema Facility to Support Data Base Update", *Proceedings of International Conference on Data Bases: Improving Usability and Responsiveness*, Haifa, Israel, 2-4 August 1978.
<MUV, NI>

[Codasyl 1962]
Codasyl Development Committee, Language Structures Group, "An Information Algebra", *Communications of the ACM*, Volume 5, Number 4, April 1962.
<DM>

[Codasyl 1971]
Codasyl Committee on Data System Languages, *Codasyl Data Base Task Group Report*, ACM, New York NY, 1971.
<NDM>

[Codasyl 1976]
Codasyl Systems Committee, *The Selection and Acquisition of Data Base Management Systems*, ACM, New York NY, March 1976.
<GI, UR>

[Codd 1969]
Codd, E. F., *Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks*, IBM Research Report RJ599, San Jose CA, August 1969.
<DM, FD, IR, RDM, SI>

[Codd 1970]
Codd, E. F., "A Relational Model for Large Shared Data Banks", *Communications of the ACM*, Volume 13, Number 6, June 1970.
<FD, RDM>

[Codd 1971a]
Codd, E. F., "A Data Base Sublanguage Founded on the Relational Calculus", *Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control*, San Diego CA, 1971.
<NI, RDM>

[Codd 1971b]
Codd, E. F., "Further Normalization of the Data Base Relational Model", *Data Base Systems* (editor Rustin, R.), Prentice Hall, 1971.
<FD, LD, RDM>

[Codd 1971c]
Codd, E. F., "Normalized Data Base Structure: A Brief Tutorial", *Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control*, San Diego CA, 1971.
<FD, LD, RDM>

[Codd 1971d]
Codd, E. F., "Relational Completeness of Data Base Sublanguages", *Data Base Systems* (editor Rustin, R.), Prentice Hall, 1971.
<NI, RDM>

[Codd 1974a]
Codd, E. F., "Recent Investigations in Relational Data Base Systems", *Information Processing '74*, North Holland, 1974.
<GI, RDM>

[Codd 1974b]
Codd, E. F., "Seven Steps to Rendezvous with the Casual User", *Data Base Management* (editors Klimbie, J. W. and K. L. Koffeman), North Holland, 1974.
<NLI, RDM>

[Codd 1974c]
Codd, E. F. and C. J. Date, "Interactive Support for Non-Programmers: The Relational and Network Approaches", *Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control*, Ann Arbor MI, 1-3 May 1974.
<NDM, NI, RDM>

[Codd 1975a]
Codd, E. F., *A List of References Pertaining to Relational Data Base Management*, IBM Research Laboratory, San Jose CA, 1975.
<RDM>

[Codd 1975b]
Codd, E. F. (editor), "Implementation of Relational Data Base Management Systems" (Transcription of 1975 National Computer Conference Panel Discussion on Relational Data Base Management), *FDT* (Bulletin of ACM SIGMOD), Volume 7, Number 2, September 1975.
<I, RDM>

[Codd 1978a]
Codd, E. F., R. S. Arnold, J. M. Cadiou, C. L. Chang, and N. Roussopoulos, *RENDEZVOUS Version 1: An Experimental English-Language Query Formulation System for Casual Users of Relational Data Bases*, IBM Research Report RJ2144, San Jose CA, 26 January 1978.
<NLI, RDM>

[Codd 1978b]
Codd, E. F., "How About Recently?", *Proceedings of International Conference on Data Bases: Improving Usability and Responsiveness*, Haifa, Israel, 2-4 August 1978.
<NLI, RDM>

[Conway 1974]
Conway, R. W., W. L. Maxwell, and H. L. Morgan, "A Technique for File Surveillance", *Information Processing '74*, North Holland, 1974.
<SI>

[Conway 1976]
Conway, R. and D. Strip, "Selective Partial Access to a Database" *Proceedings of ACM National Conference*, Houston TX, 20-22 October 1976.
<NI>

[Cook 1975]
Cook, T. J., "A Data Base Management System Design Philosophy", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, San Jose CA, 14-16 May 1975.
<LD>

[Copeland 1974]
Copeland, G. P. and S. Y. W. Su, "A High Level Data Sublanguage for a Context-Addressed Segment-Sequential Memory", *Proceedings of ACM SIGMOD Workshop on Data Description, Access, and Control*, Ann Arbor MI, May 1974.
<NI>

[Cullinane 1975a]
Cullinane Corporation, *Integrated Database Management System (IDMS): Data Definition Languages, Utilities and GCI Reference Guide*, 1975.
<I, NDM>

[Cullinane 1975b]
Cullinane Corporation, *Integrated Database Management System (IDMS): Data Manipulation Language Programmer's Reference Guide*, 1975.
<I, NDM>

[Curtice 1974]
Curtice, R. M., "Some Tools for Data Base Development", *Datamation*, Volume 20, Number 7, Pages 102-106, July 1974.
<A, GI, UR>

[Curtice 1978]
Curtice, R. M. and P. E. Jones, Jr., "Key Steps in the Logical Design of Databases", *Proceedings of NYU Symposium on Database Design*, New York NY, 18-19 May 1978.
<LD>

[Czarnik 1975]
Czarnik, B., S. Schuster, and D. Tsichritzis, "ZETA: A Relational Data Base Management System", *Proceedings of ACM Pacific Conference*, San Francisco CA, 17-18 April 1975.
<I, RDM>

[Dale 1977a]
Dale, A. G. and N. B. Dale, "Main Schema - External Schema Interaction in Hierarchically Organized Data Bases", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Toronto, Canada, 3-5 August 1977.
<HDM, LD, MUV>

[Dale 1977b]
Dale, A. G., "A Processing Interface for Multiple External Schema Access to a Data Base Management System", *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
<MUV>

[Dana 1972]
Dana, C. and L. Presser, "An Information Structure for Data Base and Device Independent Report Generation", *Proceedings of Fall Joint Computer Conference*, 1972.
<A, NI>

[Datapro 1972a]
Datapro Research Corporation, "System 2000 - MRI Systems Corporation", *Datapro 70*, April 1972.
<HDM, I>

[Datapro 1972b]
Datapro Research Corporation, "TOTAL - Cincom Systems, Inc.", *Datapro 70*, December 1972.
<I, NDM>

[Datapro 1973]
Datapro Research Corporation, "ADABAS - Software AG", *Datapro 70*, April 1973.
<HDM, I, NDM>

[Date 1971a]
Date, C. J. and P. Hopewell, "File Definition and Logical Data Independence", *Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control*, San Diego CA, 1971.
<LD>

[Date 1971b]
Date, C. J. and P. Hopewell, "Storage Structure and Physical Data Independence", *Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control*, San Diego CA, 1971.
<LD>

[Date 1972]
Date, C. J., "Relational Data Base Systems: A Tutorial", *Proceedings of Fourth Annual Symposium on Computers and Information Science*, Miami Beach FL, 14-16 December 1972.
<RDM>

[Date 1974]
Date, C. J. and E. F. Codd, "The Relational and Network Approaches: Comparison of the Application Programming Interfaces", *Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control*, Ann Arbor MI, 1-3 May 1974.
<NDM, PI, RDM>

[Date 1976]
Date, C. J., "An Architecture for High-Level Language Database Extensions", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Washington D.C., 2-4 June 1976.
<PI>

[Date 1977]
Date, C. J., *An Introduction to Data Base Systems (Second Edition)*, Addison-Wesley, 1977.
<DM, GI, HDM, NDM, NI, PI, RDM>

[Davenport 1976]
Davenport, R. A., "Database Integrity", *The Computer Journal*, Pages 110-116, May 1976.
<SI>

[Dayal 1978]
Dayal, U., "On the Updatability of Relational Views", *Proceedings of International Conference on Very Large Data Bases*, West Berlin, West Germany, 13-15 September 1978.
<MUV, RDM>

[DeBlasis 1977]
DeBlais, J. P. and T. H. Johnson, "Data Base Administration - Classical Pattern, Some Experiences and Trends", *Proceedings of National Computer Conference*, Dallas TX, 13-16 June 1977.
<GI, LD>

[DeBlasis 1978]
DeBlais, J. P. and T. H. Johnson, "Review of Data Base Administrators Functions from a Survey", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Austin TX, 31 May - 2 June 1978.
<GI, LD>

[Deheneffe 1974]
Deheneffe, C., H. Hennebert, and W. Paulus, "Relational Model for a Data Base", *Information Processing '74*, North Holland, 1974.
<RDM>

[Deheneffe 1976]
Deneheffe, C. and H. Hennebert, "NUL: A Navigational User's Language for a Network Structured Data Base", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Washington D. C., 2-4 June 1976.
<NDM, PI>

[deJong 1975]
deJong, S. P. and M. M. Zloof, *Application Design within the System for Business Automation*, IBM Reseach Report RC 5366, Yorktown Heights NY, 14 April 1975.
<A, NI, RDM>

[Delobel 1972]
Delobel, C., *A Theory about Data in an Information System*, IBM Reseach Report RJ964, San Jose CA, 28 January 1972.
<DM>

[Delobel 1973]
Delobel, C. and R. G. Casey, "Decomposition of a Data Base and the Theory of Boolean Switching Functions", *IBM Journal of Research and Development*, Volume 17, Number 5, Pages 374-386, 1973.
<DM>

[Dennis 1974]
Dennis, J. B., "First Version of a Data Flow Procedure Language", *Proceedings of Symposium on Programming*, University of Paris, Paris, France, 1974.
<NI>

[Dhaliwal 1977]
Dhaliwal, D. S. and B. R. Konsynski, "Data Integrity Considerations in Computer Based Accounting Systems", *Proceedings of National Computer Conference*, Seattle WA, 17-19 October 1977.
<SI>

[Dodd 1976]
Dodd, G. G., J. P. Fry, R. Godlove, H. Kepner, E. Sibley, and W. Steiger, "Data Base Directions: Next Steps" (transcript of panel discussion), *Proceedings of ACM National Conference*, Houston TX, 20-22 October 1976.
<GI>

[Dolk 1977]
Dolk, D. R. and M. E. Loomis, "A Methodology for the Design of Generalized Query Processors for CODASYL Databases", *Proceedings of COMPSAC '77*, Chicago IL, 8-11 November 1977.
<NDM, NI>

[Dominick 1975]
Dominick, W. D., *Models of Graphically Enhanced Data Base Management System Design*,

Ph.D. Thesis, Northwestern University, Evanston IL, 1975.
<LD>

[Donovan 1976]
Donovan, J. J., "Database System Approach to Management Decision Support", *ACM Transactions on Data Base Systems*, Volume 1, Number 4, Pages 344-369, December 1976.
<A, NI, RDM>

[Dorkenoo 1977]
Dorkenoo, E. C., M. Lemaitre, and M. Lemoine, "A Procedural Language for the Relational Data Base Management System 'SYNTEX'", *Information Processing '77*, North Holland, 1977.
<PI, RDM>

[Durchholz 1974]
Durchholz, R. and G. Richter, "Concepts for Data Base Management Systems", *Data Base Management* (editors Klimbie, J. W. and K. L. Koffeman), North Holland, 1974.
<DM, LD, S>

[Durchholz 1976]
Durchholz, R. and G. Richter, "Information Management Concepts (IMC) for Use with a DBMS Interface", *Modelling in Data Base Management Systems* (editor Nijssen, G. M.), North Holland, 1976.
<DM, LD, NI, S>

[Earley 1971]
Earley, J., "Towards an Understanding of Data Structures", *Communications of the ACM*, Volume 14, Pages 617-628, 1971.
<DM>

[Earley 1973]
Earley, J., "Relational Level Data Structures for Programming Languages", *Acta Informatica*, Volume 2, Number 4, 1973.
<DM>

[Earley 1976]
Earley, J., "High Level Iterators and a Method for Automatically Designing Data Structure Representation", *Journal of Computer Languages*, Volume 1, Pages 321-342, 1976.
<DM, LD>

[Earnest 1974]
Earnest, C. P., *A Comparison of the Network and Relational Data Structure Models*,

Computer Sciences Corporation, El Segundo CA, 1974.
<NDM, RDM>

[Eason 1976]
Eason, K., "Understanding the Naive Computer User", *The Computer Journal*, Volume 19, Number 1, Pages 3-7, 1976.
<UC>

[Edelberg 1974]
Edelberg, M., "Data Base Contamination and Recovery", *Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control*, Ann Arbor MI, 1-3 May 1974.
<SI>

[Ehrensberger 1977]
Ehrensberger, M., "Data Dictionary - More on the Impossible Dream", *Proceedings of National Computer Conference*, Dallas TX, 13-16 June 1977.
<GI, S>

[Ehrig 1978]
Ehrig, H., H. J. Kreowski, and H. Weber, "Abstract Specification Schemes for Data Base Systems", *Proceedings of International Conference on Very Large Data Bases*, West Berlin, West Germany, 13-15 September 1978.
<DM, S>

[Engel 1976]
Engel, H., *A Data Query Language: On Implementing a Data Information System*, Technical Report 6/76, Technical University of Berlin, Berlin, West Germany, 1976.
<NI>

[Engels 1971]
Engels, R. W., "An Analysis of the April 1971 DBTG Report", *Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control*, San Diego CA, 1971.
<NDM>

[Engels 1972]
Engels, R. W., "A Tutorial on Data Base Organization", *Annual Review in Automatic Programming*, Volume 7, Part 1, Pergamon Press, 1972.
<DM, LD>

[Eriksen 1974]
Eriksen, S., "The Data Base Concept", *Honeywell Computer Journal*, Volume 8, Number 1, 1974.

<GI>

[Eswaran 1975]
Eswaran, K. P. and D. D. Chamberlin, "Functional Specifications of a Subsystem for Database Integrity", *Proceedings of International Conference on Very Large Data Bases*, Framingham MA, 22-24 September 1975.
<RDM, SI>

[Eswaran 1976a]
Eswaran, K. P., *Specifications, Implementations, and Interactions of a Trigger Subsystem in a Relational Database System*, IBM Research Report RJ1820, San Jose CA, 11 August 1976.
<RDM, SI>

[Eswaran 1976b]
Eswaran, K. P., "Aspects of a Trigger Subsystem in an Integrated Data Base System", *Proceedings of Second International Conference on Software Engineering*, San Francisco CA, 13-15 October 1976.
<RDM, SI>

[Eswaran 1976c]
Eswaran, K. P., J. N. Gray, R. A. Lorie, and I. L. Traiger, "The Notions of Consistency and Predicate Locks in a Database System", *Communications of the ACM*, Volume 19, Number 11, Pages 624-633, November 1976.
<RDM>

[Everest 1974]
Everest, G. C., "The Futures of Database Management", *Proceedings of ACM SIGMOD Conference on Data Description, Access, and Control*, Ann Arbor MI, 1-3 May 1974.
<GI>

[Everest 1976]
Everest, G. C., "Basic Data Structure Models Explained with a Common Example", *Proceedings of Fifth Texas Symposium of Computing Systems*, Austin TX, 18-19 October 1976.
<DM>

[Fadous 1975a]
Fadous, R. Y. and J. Forsyth, "Finding Candidate Keys for Relational Data Bases", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, San Jose CA, 14-16 May 1975.
<FD, LD, RDM>

[Fadous 1975b]
Fadous, R. Y., *Mathematical Foundations for Relational Data Bases*, Ph.D. Thesis, Michigan State University, Lansing MI, 1975.
<FD, RDM>

[Fagin 1976]
Fagin, R., *Multivalued Dependencies and a New Normal Form for Relational Data Bases*, IBM Research Report RJ1812, San Jose CA, July 1976.
<FD, RDM>

[Fagin 1977a]
Fagin, R. "The Decomposition Versus the Synthetic Approach to Database Design, *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
<FD, LD, RDM>

[Fagin 1977b]
Fagin, R., "Multivalued Dependencies and a New Normal Form for Relational Databases", *ACM Transactions on Database Systems*, Volume 2, Number 3, Pages 262-278, September 1977.
<FD, RDM>

[Falkenberg 1976a]
Falkenberg, E., "A Uniform Approach to Data Base Management", *Modelling in Data Base Management Systems* (editor Nijssen, G. M.), North Holland, 1976.
<DM>

[Falkenberg 1976b]
Falkenberg, E., "Concepts for Modelling Information", *Modelling in Data Base Management Systems* (editor Nijssen, G. M.), North Holland, 1976.
<DM, LD, S>

[Fehder 1972]
Fehder, P. L., *The Representation-Independent Language*, IBM Research Report RJ1121, San Jose CA, 2 November 1972.
<NI>

[Fehder 1973]
Fehder, P. L., *The Representation-Independent Language*, IBM Research Report RJ1251, San Jose CA, 17 July 1973.
<NI>

[Fehder 1974]
Fehder, P., "HQL: A Set-Oriented Transaction Language for Hierarchically Structured Data Bases", *Proceedings of ACM National Conference*, San Diego CA, November 1974.
<HDM, NI>

[Fernandez 1975]
Fernandez, E. B., R. C. Summers, and C. D. Coleman, "An Authorization Model for a Shared Data Base", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, San Jose CA, 14-16 May 1975.
<NI>

[Fernandez 1976]
Fernandez, E. B. and R. C. Summers, "Integrity Aspects of a Shared Data Base", *Proceedings of National Computer Conference*, New York NY, 7-10 June 1976.
<SI>

[Fields 1977]
Fields, C. "Human Interfaces to Very Large Data Base Systems", *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
<NI, PI, UR>

[Fikes 1977]
Fikes, R. and G. Hendrix, "A Network-Based Knowledge Representation and Its Natural Deduction System", *Proceedings of International Joint Conference on Artificial Intelligence*, Cambridge MA, 22-25 August 1977.
<S>

[Fillmore 1968]
Fillmore, C., "The Case for Case", *Universals in Linguistic Theory* (editors Bach, E. and R. Harms), Holt, Rinehart, and Winston, 1968.
<NLI, S>

[Finin 1978]
Finin, T. W., *Casting the RENDEZVOUS Analyzer Rules in Augmented Transition Network Form*, IBM Research Report RJ2146, San Jose CA, January 1978.
<NLI, RDM>

[Florentin 1974]
Florentin, J. J., "Consistency Auditing of Databases", *The Computer Journal*, Volume 17, Number 1, February 1974.
<SI>

[Florentin 1976]
Florentin, J. J., "Information Reference Coding", *Communications of the ACM*, Volume 19, Number 1, January 1976.
<DM>

[Flory 1978]
Flory, A. and J. Kouloumdijian, "A Model and Method for Logical Database Design", *Proceedings of International Conference on Very Large Data Bases*, West Berlin, West Germany, 13-15 September 1978.
<LD>

[Forehand 1975]
Forehand, J. C. R., *A High Level Language Implementation of the Common Features of Data Base Definitions*, Ph.D. Thesis, Texas A&M University, Texas, 1975.
<LD>

[Fossum 1974]
Fossum, B. M., "Data Base Integrity as Provided for by a Particular Data Base Management System", *Data Base Management* (editors Klimbie, J. W. and K. L. Koffeman), North Holland, 1974.
<A>

[Foucaut 1978]
Foucaut, O. and C. Rolland, "Concepts for Design of an Information System - Conceptual Schema and Its Utilization in the REMORA Project", *Proceedings of International Conference on Very Large Data Bases*, West Berlin, West Germany, 13-15 September 1978.
<DM, LD>

[Franking 1977]
Franking, N. A., E. P. Nellen, A. D. Inselberg, and A. K. Olson, "Providing Data Integrity and Security Through Software Interfaces", *Proceedings of Conference on Computer Security and Integrity - Trends and Applications*, Gaithersburg MD, 19 May 1977.
<PI, SI>

[Fry 1976a]
Fry, J. P. and B. K. Kahn, *A Stepwise Approach to Database Design*, Technical Report 76 DE1, Data Translation Project, Graduate School of Business Administration, University of Michigan, Ann Arbor MI, May 1976.
<LD>

[Fry 1976b]
Fry, J. P. and E. H. Sibley, "Evolution of Data-Base Management Systems", *Computing*

Surveys, Volume 8, Number 1, March 1976.
<GI>

[Furakawa 1977]
Furakawa, K., "A Deductive Question Answering System on Relational Data Bases", *Proceedings of International Joint Conference on Artificial Intelligence*, Cambridge MA, 22-25 August 1977.
<DI, NI, RDM>

[Furtado 1977]
Furtado, A. L., "An Algebra of Quotient Relations", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Toronto, Canada, 3-5 August 1977.
<NI, RDM>

[Gambino 1977]
Gambino, T. J. and R. Gerritsen, "A Database Design Decision Support System", *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
<LD>

[Genton 1970]
Genton, A., "Recovery Procedures for Direct Access Commercial Systems", *Computer Journal*, Volume 13, Pages 123-126, 1970.
<A>

[Gerritsen 1975]
Gerritsen, R., "A Preliminary System for the Design of DBTG Data Structures", *Communications of the ACM*, Volume 18, Number 10, October 1975.
<NDM>

[Gerritsen 1978]
Gerritsen, R., "Steps Toward the Automation of Database Design", *Proceedings of NYU Symposium on Database Design*, New York NY, 18-19 May 1978.
<LD>

[Giordano 1976]
Giordano, N. J. and M. S. Schwartz, "Data Base Recovery at CMIC", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Washington D. C., 2-4 June 1976.
<A>

[Goldberg 1975]
Goldberg, P. C., "Automatic Programming", *Lecture Notes in Computer Science*, Volume 23, Springer Verlag, 1975.
<NI>

[Goldstein 1970]
Goldstein, R. C. and A. L. Strnad, "The MacAims Data Management System", *Proceedings of ACM SIGFIDET Workshop on Data Description and Access*, November 1970.
<I, RDM>

[Goodenough 1975]
Goodenough, J. B., "Structured Exception Handling", *Proceedings of Second ACM Symposium on Principles of Programming Languages*, 20-22 January 1975.
<SI>

[Gosden 1974]
Gosden, J. A., "Large Scale Data Base Systems - Current Deficiencies and User Requirements", *Data Base Management Systems* (editor Jardine, D. A.), North Holland, 1974.
<GI, UR>

[Gotlieb 1975]
Gotlieb, L. R., "Computing Joins of Relations", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, San Jose CA, 14-16 May 1975.
<FD, RDM>

[Grandwell 1975]
Grandwell, J. L., "Why Data Dictionaries?", *Database*, Volume 6, Number 2, Pages 15-18, 1975.
<DM, LD, S, UR>

[Graves 1975]
Graves, R. W., "Integrity Control in a Relational Data Description Language", *Proceedings of ACM Pacific Conference*, San Francisco CA, 17-18 April 1975.
<RDM, SI>

[Green 1969]
Green, C. C., "Theorem-Proving by Resolution as a Basis for Question-Answering Systems", *Machine Intelligence* (editors Meltzer, B. and D. Michie), 1969.
<DI>

[Greenblatt 1978]
Greenblatt, D. and J. Waxman, "A Study of Three Database Query Languages",

Proceedings of International Conference on Data Bases: Improving Usability and Responsiveness, Haifa, Israel, 2-4 August 1978.
<NI>

[Griffiths 1976]
Griffiths, P. P. and B. W. Wade, "An Authorization Mechanism for a Relational Data Base System", *ACM Transactions on Database Systems*, Volume 1, Number 3, Pages 242-255, September 1976.
<RDM>

[Grossman 1975]
Grossman, R. W., *Representing the Semantics of Natural Language as Constraint Expressions*, Working Paper 87, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge MA, January 1975.
<S>

[Grossman 1976]
Grossman, R. W., *Some Data Base Applications of Constraint Expressions*, Technicial Report TR-158, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge MA, February 1976.
<S>

[Guttag 1976a]
Guttag, J., "Abstract Data Types and the Development of Data Structures", *Proceedings of ACM SIGPLAN/SIGMOD Conference on Data: Abstraction, Definition, and Structure*, Salt Lake City UT, 22-24 March 1976.
<DM, LD>

[Guttag 1976b]
Guttag, J., E. Horowitz, and D. Muser, "The Design of Data Structure Specifications", *Proceedings of Second International Conference on Software Engineering*, San Francisco CA, 13-15 October 1976.
<DM, LD>

[Hainaut 1974]
Hainaut, J. L. and B. Lechartier, "An Extensible Semantic Model of Data Bases and Its Data Language", *Information Processing '74*, North Holland, 1974.
<DM, NI, S>

[Hall 1975]
Hall, P., P. Hitchcock, and S. Todd, "An Algebra of Relations for Machine Computation, *Proceedings of Second ACM Symposium on Principles of Programming Languages*, 20-22

January 1975.
<NI, RDM>

[Hall 1976]
Hall, P., J. Owlett, and S. Todd, "Relations and Entities", *Modelling in Data Base Management Systems* (editor Nijssen, G. M.), North Holland, 1976.
<RDM, S>

[Hammer 1974]
Hammer, M. M., W. G. Howe, and I. Wladawsky, *An Interactive Business Definition System*, IBM Research Report RC4680, Yorktown Heights NY, 16 January 1974.
<A, NI>

[Hammer 1975a]
Hammer, M. M. and D. J. McLeod, "Semantic Integrity in a Relational Data Base System", *Proceedings of International Conference on Very Large Data Bases*, Framingham MA, 22-24 September 1975.
<RDM, SI>

[Hammer 1975b]
Hammer, M. M., "The Design of Usable Programming Languages", *Proceedings of ACM National Conference*, Minneapolis MN, 20-22 October 1975.
<NI, PI>

[Hammer 1975c]
Hammer, M. M., W. G. Howe, V. J. Kruskal, and I. Wladawsky, *A Very High Level Programming Language for Data Processing Applications*, IBM Research Report RC5583, Yorktown Heights NY, 15 August 1975.
<NI>

[Hammer 1976a]
Hammer, M. M. and A. Y. Chan, "Index Selection in a Self-Adaptive Data Base Management System", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Washington D. C., 2-4 June 1976.
<RDM>

[Hammer 1976b]
Hammer, M. M. and A. Y. Chan, "Acquisition and Utilization of Access Patterns in Relational Data Base Implementation", *Pattern Recognition and Artificial Intelligence* (editor Chen, C. H.), Academic Press, 1976.
<NI, RDM>

[Hammer 1976c]
Hammer, M. M., "Error Detection in Data Base Systems", *Proceedings of National Computer Conference*, New York NY, 7-10 June 1976.
<SI>

[Hammer 1976d]
Hammer, M. M. and D. J. McLeod, "A Framework for Data Base Semantic Integrity", *Proceedings of Second International Conference on Software Engineering*, San Francisco CA, 13-15 October 1976.
<RDM, SI>

[Hammer 1976e]
Hammer, M. M., "Data Abstractions for Data Bases", *Proceedings of ACM SIGMOD/SIGPLAN Conference on Data: Abstraction, Definition, and Structure*, Salt Lake City UT, 22-24 March 1976.
<DM, LD, S>

[Hammer 1977a]
Hammer, M., W. G. Howe, V. J. Kruskal, and I. Wladawsky, "A Very High Level Programming Language for Data Processing Applications", *Communications of the ACM*, Volume 20, Number 11, Pages 832-840, November 1977.
<A, NI>

[Hammer 1977b]
Hammer, M., "Self-Adaptive Automatic Database Design", *Proceedings of National Computer Conference*, Dallas TX, 13-16 June 1977.
<PI, RDM>

[Hammer 1978]
Hammer, M. and D. McLeod, "The Semantic Data Model: A Modelling Mechanism for Data Base Applications", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Austin TX, 31 May - 2 June 1978.
<DM, IR, LD, NI, S>

[Hardgrave 1976]
Hardgrave, W. T., "Set Procesing: A Tool for Data Management", *Proceedings of ACM/NBS Fifteenth Annual Technical Symposium*, June 1976.
<DM>

[Harris 1977]
Harris, L. R., "User Oriented Data Base Query with the ROBOT Natural Language Query System", *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-

8 October 1977.
<NLI>

[Hawkinson 1975]
Hawkinson, L., "The Representation of Concepts in OWL", *Proceedings of International Joint Conference on Artificial Intelligence*, Tbilisi, Georgia, USSR, 3-8 September 1975.
<NLI, S>

[Hawley 1975]
Hawley, D. A., J. S. Knowles, and E. E. Tozer, "Database Consistency and the CODASYL DBTG Proposals, *The Computer Journal*, Volume 16, Number 3, November 1975.
<NDM, SI>

[Hawryskiewycz 1972]
Hawryskiewycz, I. T. and J. B. Dennis, "An Approach to Proving the Correctness of Data Base Operations", *Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control*, November 1972.
<RDM>

[Hawryskiewycz 1973]
Hawryskiewycz, I. T., *Semantics of Data Base Systems*, Technical Report TR-112, Project MAC, Massachusetts Institute of Technology, Cambridge MA, December 1973.
<RDM>

[Hayes 1977]
Hayes, P. J., "On Semantic Nets, Frames, and Association", *Proceedings of International Joint Conference on Artificial Intelligence*, Cambridge MA, 22-25 August 1977.
<S>

[Heath 1971]
Heath, I. J., "Unacceptable File Operations in a Relational Data Base", *Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control*, San Diego CA, 1971.
<NI, RDM, S>

[Helbig 1977]
Helbig, H., "A New Method for Deductive Answer Finding in a Question-Answering System", *Information Processing '77*, North Holland, 1977.
<DI, NI>

[Held 1975a]
Held, G., M. R. Stonebraker, and E. Wong, "INGRES: A Relational Data Base System", *Proceedings of National Computer Conference*, Anaheim CA, 19-22 May 1975.

<I, NI, RDM>

[Held 1975b]
Held, G. and M. Stonebraker, "Networks, Hierarchies, and Relations in Data Base Management Systems", *Proceedings of ACM Pacific Conference*, San Francisco CA, 17-18 April 1975.
<HDM, NDM, NI, RDM>

[Hendrix 1977a]
Hendrix, G. G., "LIFER: A Natural Language Interface Facility", *Proceedings of the Second Berkeley Workshop on Distributed Data Management and Computer Networks*, Berkeley CA, 25-27 May 1977.
<NLI>

[Hendrix 1977b]
Hendrix, G. G., E. D. Sacerdoti, D. Sagalowicz, and J. Slocum, "Developing a Natural Language Interface to Computer Data", *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
<NLI>

[Hendrix 1977c]
Hendrix, G. G., "Human Engineering for Applied Natural Language Processing", *Proceedings of International Joint Conference on Artificial Intelligence*, Cambridge MA, 22-25 August 1977.
<NLI>

[Hendrix 1978]
Hendrix, G. G., E. D. Sacerdoti, D. Sagalowicz, and J. Slocum, "Developing a Natural Language Interface to Complex Data", *ACM Transactions on Database Systems*, June 1978.
<NLI>

[Hewitt 1971]
Hewitt, C. E., "Procedural Embedding of Knowledge in PLANNER", *Proceedings of International Joint Conference on Artificial Intelligence*, September 1971.
<DI>

[Hitchcock 1974]
Hitchcock, P., *Fundamental Operations on Relations in a Relational Data Base*, IBM Scientific Centre Report UKSC-0051, Peterlee, England, May 1974.
<NI, RDM>

[Hitchcock 1976]
Hitchcock, P., "User Extensions to the Peterlee Relational Test Vehicle", *Systems for Large Data Bases* (editors Lockemann, P. C. and E. J. Neuhold), North Holland, 1976.
<I, MUV, RDM>

[Honeywell 1972]
Honeywell Information Systems, *Integrated Data Store Reference Manual*, Order Number BR69, Wellesley MA, 1972.
<I, NDM>

[Hotaka 1977]
Hotaka, R. and M. Tsubaki, "Self-Descriptive Relational Data Bases", *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
<RDM, S>

[Housel 1975]
Housel, B. C., D. P. Smith, N. C. Shu, and V. Y. Lum, "Data Translation, Part II: DEFINE: A Nonprocedural Data Description Language for Defining Information Easily", *Proceedings of ACM Pacific Conference*, San Francisco CA, April 1975.
<DM>

[Housel 1976]
Housel, B. C. and N. C. Shu, "A High Level Manipulation and Query Language for Hierarchical Data Abstractions", *Proceedings of ACM SIGPLAN/SIGMOD Conference on Data: Abstraction, Definition, and Structure*, Salt Lake City UT, 22-24 March 1976.
<HDM, NI>

[Housel 1977]
Housel, B., "A Unified Approach to Program and Data Conversion", *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
<DM, PI>

[Hsiao 1975]
Hsiao, D. K., *Systems Programming Concepts of Operating and Data Base Systems*, Addison Wesley, 1975.
<DM, GI>

[Hsiao 1976]
Hsiao, D., "A Software Engineering Experience in the Management, Design, and Implementation of a Data - Secure System", *Proceedings of Second International Conference on Software Engineering*, San Francisco CA, 13-15 October 1976.
<A>

[Hubbard 1975]
Hubbard, G. and N. Raver, "Automating Logical File Design", *Proceedings of International Conference on Very Large Data Bases*, Framingham MA, 22-24 September 1975.
<LD>

[Hubbard 1978]
Hubbard, G. U., "Techniques for Automated Logical Database Design", *Proceedings of NYU Symposium on Database Design*, New York NY, 18-19 May 1978.
<LD>

[Huits 1975]
Huits, M., "Requirements for Languages in Data Base Systems", *Data Base Description* (editors Douque, B. C. M. and G. M. Nijssen), North Holland, 1975.
<NI, PI, UR>

[IBM 1975a]
IBM, *Information Management System Virtual Storage (IMS/VS) General Information Manual*, Form Number GH20-1260-3, White Plains NY, 1975.
<HDM, I>

[IBM 1975b]
IBM, *Information Management System Virtual Storage (IMS/VS) System/Application Design Guide*, Form Number SH20-9025-2, White Plains NY, 1975.
<HDM, I>

[Inselberg 1976]
Inselberg, A., A. Olson, and N. Franking, *DTL: A Data Transformation Language to Provide Data Base Integrity*, Technical Report, Intel, Santa Clara CA, 1976.
<PI, SI>

[Jackendoff 1975]
Jackendoff, R., "A System of Semantic Primitives", *Proceedings of Workshop in the Theoretical Issues of Natural Language Processing*, Cambridge MA, 1975.
<NLI, S>

[Jervis 1974]
Jervis, B. M., *Query Languages for Relational Data Base Management Systems*, S. M. Thesis, Department of Computer Science, University of British Columbia, Canada, May 1974.
<NI, RDM>

[Joyce 1974]
Joyce, J. D., J. T. Murray, and M. R. Ward, "Data Management System User Requirements", *Data Base Management Systems* (editor Jardine, D. A.), North Holland, 1974.
<UR>

[Kahn 1976]
Kahn, B. K., "A Method for Describing the Information Required by the Database Design Process", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Washington D.C., 2-4 June 1976.
<LD>

[Kahn 1978]
Kahn, B., "A Structured Logical Database Design Methodology", *Proceedings of NYU Symposium on Database Design*, New York NY, 18-19 May 1978.
<LD>

[Kambayashi 1977]
Kambayashi, Y., K. Tanaka, and S. Yajima, "A Relational Data Language with Simplified Binary Relation Handling Capability", *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
<NI, RDM>

[Kameny 1978]
Kameny, I, "EUFID: The End-User Friendly Interface to Data Management Systems", *Proceedings of International Conference on Very Large Data Bases*, West Berlin, West Germany, 13-15 September 1978.
<NLI>

[Katzan 1975]
Katzan, H., *Computer Data Management and Data Base Technology*, Van Nostrand Reinhold, 1975.
<DM, GI>

[Kay 1975]
Kay, M. H., "An Assessment of the CODASYL DDL for Use with a Relational Schema", *Data Base Description* (editors Douque, B. C. M. and G. M. Nijssen), North Holland, 1975.
<NDM, RDM>

[Kayel 1977]
Kayel, R. G., *An Approach to Data Management which Separates Structure from Data*, Technical Report, Bell Telephone Laboratories, Murray Hill NJ, 1977.
<DM, LD>

[Kellogg 1971]
Kellogg, C. H., J. Burger, T. Diller, and K. Fogt, "The CONVERSE Natural Language Data Management System: Current Status and Plans", *Proceedings of ACM Symposium on Information Storage and Retrieval*, College Park MD, 1971.
<I, NLI>

[Kellogg 1976]
Kellogg, C., P. Klahr, and L. Travis, "A Deductive Capability for Data Management", *Systems for Large Data Bases* (editors Lockemann, P. C. and E. J. Neuhold), North Holland, 1976.
<DI>

[Kellogg 1977]
Kellogg, C., P. Klahr, and L. Travis, "Deductive Methods for Large Data Bases", *Proceedings of International Joint Conference on Artificial Intelligence*, Cambridge MA, 22-25 August 1977.
<DI>

[Kent 1973]
Kent, W., *A Primer on Normal Forms*, IBM System Development Division Technical Report TR02.600, San Jose CA, 17 December 1973.
<FD, RDM>

[Kent 1976]
Kent, W., "New Criteria for the Conceptual Model", *Systems for Large Data Bases* (editors Lockemann, P. C. and E. J. Neuhold), North Holland, 1976.
<DM, LD>

[Kerschberg 1976a]
Kerschberg, L., A. Klug, and D. Tsichritzis, *A Taxonomy of Data Models*, Technical Report CSRG-70, Computer Systems Research Group, University of Toronto, Canada, May 1976.
<DM>

[Kerschberg 1976b]
Kerschberg, L., E. A. Ozkarahan, and J. E. S. Pacheco, "A Synthetic English Query Language for a Relational Associative Processor", *Proceedings of Second International Conference on Software Engineering*, San Francisco CA, 13-15 October 1976.
<I, NI, RDM>

[Kerschberg 1976c]
Kerschberg, L., D. Tsichritzis, and A. Klug, "A Taxomony of Data Models", *Systems for*

Large Data Bases (editors Lockemann, P. C. and E. J. Neuhold), North Holland, 1976.
<DM>

[Kerschberg 1976d]
Kerschberg, L. and J. E. S. Pacheco, *A Functional Data Base Model*, Technical Report 13, Department of Information Systems Management, University of Maryland, College Park MD, 1976.
<DM, FD>

[King 1973]
King, P. F. and J. E. Shemer, "ARS - An Interactive Reporting System", *Proceedings of ACM SIGPLAN-SIGIR Interface Meeting*, Gaithersburg MD, 4-6 November 1973.
<A>

[Kleefstra 1978]
Kleefstra, W. J., "Database Description with a Single Name Category Data Model", *Proceedings of International Conference on Very Large Data Bases*, West Berlin, West Germany, 13-15 September 1978.
<DM, LD, S>

[Klug 1977]
Klug, A. and D. Tsichritzis, "Multiple View Support within the ANSI/X3/SPARC Framework", *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
<DM, MUV>

[Kobayashi 1975]
Kobayashi, I., "Information and Information Processing Structure", *Information Systems*, Volume 1, Pages 39-49, 1975.
<GI>

[Koss 1976]
Koss, A. M. and J. M. Noonan, "A Case Study in the Use of Data Base Management", *Proceedings of ACM National Conference*, Houston TX, 20-22 October 1976.
<A>

[Kraegeloh 1973]
Kraegeloh, K. P. and P. C. Lockemann, "Retrieval in Set-Theoretically Stored Data Bases: Concepts and Practical Considerations", *Proceedings of International Computing Symposium*, North Holland, 1973.
<NI, RDM>

[Krauss 1952]
Krauss, R. *A Hole is to Dig*, Harper and Row, 1952.
<GI>

[Kroenke 1977]
Kroenke, D., *Database Processing*, Science Research Associates, 1977.
<GI, NDM>

[Kunii 1976]
Kunii, T. L., "DATAPLAN: An Interface Generator for Database Semantics", *Information Systems*, Volume 10, Pages 279-298, 1976.
<NI, S>

[Lacroix 1975]
Lacroix, M., *An English Structured Query Language: ILL*, Technical Report R301, M.B.L.E. Research Laboratory, Brussels, Belgium, July 1975.
<NI, RDM>

[Lacroix 1976]
Lacroix, M., *Example Queries in Relational Languages*, Technical Report N107, M.B.L.E. Research Laboratory, Brussels, Belgium, January 1976.
<NI, RDM>

[Lacroix 1977a]
Lacroix, M., *ILL: An English Structured Query Language for Relational Data Bases*, Technical Report R334, M.B.L.E. Research Laboratory, Brussels, Belgium, January 1977.
<NI, RDM>

[Lacroix 1977b]
Lacroix, M. and A. Pirotte, "Domain-Oriented Relational Languages", *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
<NI, RDM>

[Lang 1977]
Lang, T., E. B. Fernandez, and R. C. Summers, "A System Architecture for Compile-Time Actions in Databases", *Proceedings of National Computer Conference*, Seattle WA, 17-19 October 1977.
<SI>

[Langefors 1974]
Langefors, B., "Information Systems", *Information Processing '74*, North Holland, 1974.
<DM, S>

[Langefors 1977]
Langefors, B., "Information Systems Theory", *Information Systems*, Volume 2, Pages 207-219, 1977.
<DM, S>

[Leavenworth 1975]
Leavenworth, B. M., *Non-Procedural Data Processing*, IBM Research Report RC5417, Yorktown Heights NY, 14 May 1975.
<NI>

[Ledgard 1977]
Ledgard, H. F. and R. W. Taylor, "Two Views of Data Abstraction", *Communicatons of the ACM*, Volume 20, Number 6, Pages 382-384, June 1977.
<DM, LD>

[Lee 1976]
Lee, R. C. T., J. R. Slagle, and C. T. Wong, "Application of Clustering to Estimate Missing Data and Improve Data Integrity", *Proceedings of Second International Conference on Software Engineering*, San Francisco CA, 13-15 October 1976.
<SI>

[Lee 1978a]
Lee, R. M. and R. Gerritsen, "Extended Semantics for Generalization Hierarchies", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Austin TX, 31 May - 2 June 1978.
<DM, LD, S>

[Lee 1978b]
Lee, R. M., "Conversational Aspects of Database Interactions", *Proceedings of International Conference on Very Large Data Bases*, West Berlin, West Germany, 13-15 September 1978.
<NI, NLI>

[Leech 1974]
Leech, G., *Semantics*, Penguin Books, 1974.
<S>

[Levien 1967]
Levien, R. E. and M. E. Maron, "A Computer System for Inference Execution and Data Retrieval", *Communications of the ACM*, Volume 10, Number 11, November 1967.
<DI, NI>

[Lewis 1977a]
Lewis, E. A., L. C. Sekino, and R. D. Ting, *Data Semantics and Database Update Rules Based on Functional Dependencies*, Technical Report, Bell Telephone Laboratories, Holmdel NJ, 1977.
<FD, S>

[Lewis 1977b]
Lewis, E. A., L. C. Sekino, and P. D. Ting, "A Canonical Representation for the Relational Schema and Logical Data Independence", *Proceedings of COMPSAC '77*, Chicago IL, 8-11 November 1977.
<LD, RDM>

[Lien 1977]
Lien, Y. E., "Design and Implementation of a Relational Database on a Minicomputer", *Proceedings of National Computer Conference*, Seattle WA, 17-19 October 1977.
<I, RDM>

[Lindgreen 1974]
Lindgreen, P., "Basic Operations on Information as a Basis for Data Base Design", *Information Processing '74*, North Holland, 1974.
<DM, LD>

[Lipski 1977]
Lipski, W., "On Semantic Issues Connected with Incomplete Information Data Bases", *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
<S>

[Liskov 1974]
Liskov, B. and S. Zilles, "Programming with Abstract Data Types", *Proceedings of a Symposium on Very High Level Languages*, Santa Monica CA, March 1974.
<DM, S>

[Liskov 1977]
Liskov, B. H., A. Snyder, R. Atkinson, and C. Schaffert, "Abstraction Mechanisms in CLU", *Communications of the ACM*, Volume 12, Number 8, Pages 564-576, November 1977.
<DM, S>

[Liskov 1978]
Liskov, B., E. Moss, C. Schaffert, B. Scheifler, and A. Snyder, *CLU Reference Manual*, Computation Structures Group Memo 161, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge MA, July 1978.

<DM, S>

[Lochovsky 1976a]
Lockovsky, F. H. and D. C. Tsichritzis, *Human Factors Considerations in DBMS Selection*, Technical Report, Department of Computer Science, University of Toronto, Toronto, Canada, 1976.
<UC>

[Lochovsky 1976b]
Lochovsky, F. H. and D. C. Tsichritzis, "An Educiational Data Base Management System", *Infor*, Volume 14, Number 3, Pages 270-278, October 1976.
<HDM, I, NDM, RDM>

[Lochovsky 1976c]
Lochovsky, F. H., *User Performance Measures for Data Base Management Systems*, Technical Report, Department of Computer Science, University of Toronto, Toronto, Canada, 1976.
<UC>

[Lochovsky 1977]
Lockovsky, F. H. and D. C. Tsichritzis, "User Performance in Data Base Management System Selection", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Toronto, Canada, 3-5 August 1977.
<UC>

[Lorie 1971]
Lorie, R. A. and A. J. Symonds, "A Relational Access Method for Interactive Applications", *Data Base Systems* (editor Rustin, R.), Prentice Hall, 1971.
<NI, RDM>

[Lorie 1974]
Lorie, R. A., *XRM - An Extended (N-ary) Relational Memory*, IBM Scientific Center Technical Report G320-2096, Cambridge MA, January 1974.
<I, RDM>

[Lozinski 1976]
Lozinski, E. L., "Relations, Transformations, and Redundancy in Relational Data Base Systems", *Systems for Large Data Bases* (editors Lockemann, P. C. and E. J. Neuhold), North Holland, 1976.
<IR, RDM, S>

[Lyon 1971]
Lyon, J. K., *An Introduction to Data Base Design*, John Wiley and Sons, 1971.
<LD>

[Lyon 1976]
Lyon, J. K., *The Database Administrator*, John Wiley and Sons, 1976.
<LD, UR>

[Machgeels 1976]
Machgeels, C., "A Procedural Language for Expressing Integrity Constraints in the Coexistence Model", *Modelling in Data Base Management Systems* (editor Nijssen, G. M.), North Holland, 1976.
<DM, PI, SI>

[Makinouchi 1977]
Makinouchi, A., "A Consideration on Normal Form of Not-Necessarily-Normalized Relations in the Relational Data Model", *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
<FD, LD, RDM>

[Malkin 1977]
Malkin, J. G. and B. F. Anderson, "Rehabilitation Information System: Three Views of an Application", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Toronto, Canada, 3-5 August 1977.
<A>

[Manola 1978]
Manola, F., "A Review of the 1978 CODASYL Database Specifications", *Proceedings of International Conference on Very Large Data Bases*, West Berlin, West Germany, 13-15 September 1978.
<NDM>

[Marill 1975]
Marill, T. and D. Stern, "The Datacomputer: A Network Utility", *Proceedings of National Computer Conference*, Anaheim CA, 19-22 May 1975.
<HDM, I>

[Martin 1975]
Martin, J. T., *Computer Data-Base Orgranization*, Prentice Hall, 1975.
<DM, GI, NI, PI>

[Martin 1976]
Martin, J. T., *Principles of Data-Base Management*, Prentice Hall, 1976.
<DM, GI, HDM, NDM, NI, PI, RDM>

[Mason 1978]
Mason, P. J., "A DBMS Architecture to Support Information Analysis", *Proceedings of International Conference on Very Large Data Bases*, West Berlin, West Germany, 13-15 September 1978.
<LD, UR>

[Maynard 1974]
Maynard, H. S., "User Requirements for Data Base Management Systems (DBMS)", *Data Base Management Systems* (editor Jardine, D. A.), North Holland, 1974.
<UR>

[McCrea 1976]
McCrea, D. R., "The Programming Language - Data Base Management System Interface in the BIS/3000 System", *Proceedings of Fifth Texas Conference on Computing Systems*, Austin TX, 18-19 October 1976.
<PI>

[McDonald 1974a]
McDonald, N., M. Stonebraker, and E. Wong, *Preliminary Design of INGRES Part I - Query Language, Data Storage and Access*, Electronics Research Laboratory Report ERL-M435, University of California, Berkeley CA, 10 April 1974.
<NI, RDM>

[McDonald 1974b]
McDonald, N. M., M. Stonebraker, and E. Wong, *Preliminary Design of INGRES Part II - Protection, Concurrency and Graphics*, Electronics Research Laboratory Report ERL-M436, University of California, Berkeley CA, 9 May 1974.
<NI, RDM>

[McDonald 1975a]
McDonald, N. and M. Stonebraker, "CUPID: The Friendly Query Language", *Proceedings of ACM Pacific Conference*, San Francisco CA, 17-18 April 1975.
<NI, RDM>

[McDonald 1975b]
McDonald, N. H., *CUPID: A Graphics Oriented Facility for Support of Non-Programmer Interactions with a Data Base*, Electronics Research Laboratory Report ERL-M563, University of California, Berkeley CA, 12 November 1975.

<NI, RDM, UC>

[McDonald 1975c]
McDonald, N., *Getting Started in INGRES with CUPID: A Tutorial*, Electronics Research Laboratory Report ERL-M546, University of California, Berkeley CA, 1 September 1975.
<NI, RDM>

[McGee 1972]
McGee, W. C., "Some Current Issues in Data Description", *Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control*, November 1972.
<DM, GI, LD>

[McGee 1974]
McGee, W. C., "A Contribution to the Study of Data Equivalence", *Data Base Management* (editors Klimbie, J. W. and K. L. Koffeman), North Holland, 1974.
<DM>

[McGee 1975]
McGee, W. C., "File-Level Operations on Network Data Structures, *Proceedings of ACM SIGMOD International Conference on the Management of Data*, San Jose CA, 14-16 May 1975.
<NDM, PI>

[McGee 1976]
McGee, W. C., "On User Criteria for Data Model Evaluation", *ACM Transactions on Data Base Systems*, Volume 1, Number 4, Pages 370-387, December 1976.
<DM, UC, UR>

[McLeod 1974]
McLeod, D. J., *Relational Data Management in Minicomputers*, S. B. Thesis, Department of Electrical Engineering, Massachusetts Institute of Technology, Cambridge MA, February 1974.
<I, NI, PI, RDM>

[McLeod 1975]
McLeod, D. J. and M. J. Meldman, "RISS: A Generalized Minicomputer Relational Data Base Management System", *Proceedings of National Computer Conference*, Anaheim CA, 19-22 May 1975.
<I, NI, PI, RDM>

[McLeod 1976a]
McLeod, D. J., *High Level Domain Definition in a Relational Data Base System*, IBM

Research Report RJ1716, San Jose CA, 9 February 1976.
<RDM, SI>

[McLeod 1976b]
McLeod, D. J., "High Level Domain Definition in a Relational Data Base System", *Proceedings of ACM SIGPLAN/SIGMOD Conference on Data: Abstraction, Definition, and Structure*, Salt Lake City UT, 22-24 March 1976.
<RDM, SI>

[McLeod 1976c]
McLeod, D. J., *Query by Example and SEQUEL: Translation and Compatibility*, IBM Research Report RJ1730, San Jose CA, 27 February 1976.
<NI, RDM>

[McLeod 1976d]
McLeod, D. J., "The Translation and Compatibility of SEQUEL and Query by Example", *Proceedings of the Second International Conference on Software Engineering*, San Francisco CA, 13-15 October 1976.
<NI, RDM>

[McLeod 1976e]
McLeod, D. J., *High Level Expression of Semantic Integrity Specifications in a Relational Data Base System*, Technical Report TR-165, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge MA, September 1976.
<RDM, SI>

[McLeod 1977a]
McLeod, D. J., "High Level Definition of Abstract Domains in a Relational Data Base System", *Journal of Computer Languages*, Volume 2, Number 3, 1977.
<RDM, SI>

[McLeod 1977b]
McLeod, D., "A Framework for Data Base Protection and Its Application to the INGRES and System R Data Base Management Systems", *Proceedings of COMPSAC '77*, Chicago IL, 8-11 November 1977.
<RDM>

[Mealey 1967]
Mealey, G. H., "Another Look at Data", *Proceedings of Fall Joint Computer Conference*, 1967.
<GI>

[Meldman 1978]
Meldman, M. J., D. J. McLeod, R. J. Pellicore, and M. B. Squire, *RISS: A Relational Data Base Management System for Minicomputers*, Van Nostrand Reinhold, 1978.
<I, RDM>

[Meltzer 1973]
Meltzer, H. S., "Current Concepts in Data Base Design", *IBM Report to GUIDE 37 Information Systems Division*, 2 November 1973.
<LD>

[Meltzer 1976]
Meltzer, H. S., "Structure and Redundancy in the Conceptual Schema in the Administration of Very Large Data Bases", *Systems for Large Data Bases* (editors Lockemann, P. C. and E. J. Neuhold), North Holland, 1976.
<DM, IR, LD>

[Merrett 1978]
Merrett, H., "The Extended Relational Algebra, a Basis for Query Languages", *Proceedings of International Conference on Data Bases: Improving Usability and Responsiveness*, Haifa, Israel, 2-4 August 1978.
<NI, RDM>

[Merten 1976]
Merten, A. G., *A Theoretical Analysis of Data Definition and Translation*, Technical Report AFOSR-TR-76-0556, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor MI, 1976.
<DM, LD>

[Merten 1977]
Merten, A., "A Prospective on Database Design", *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
<LD>

[Metaxides 1975]
Metaxides, A., "Information Bearing and Non Information Bearing Sets", *Data Base Description* (editors Douque, B. C. M. and G. M. Nijssen), North Holland, 1975.
<IR, NDM>

[Michaels 1976]
Michaels, A. S., B. Mittman, and R. C. Carlson, "A Comparison of the Relational and CODASYL Approaches to Data-Base Management", *Computing Surveys*, Volume 8, Number 1, March 1976.

<NDM, RDM>

[Miller 1974]
Miller, L. A., "Programming by Non-Programmers", *International Journal of Man-Machine Studies*, Volume 6, Pages 237-260, 1974.
<NI>

[Minker 1975]
Minker, J. "Performing Inferences Over Relational Data Bases", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, San Jose CA, 14-16 May 1975.
<DI, RDM>

[Minsky 1974a]
Minsky, N., "On Interaction with Data Bases", *Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control*, Ann Arbor MI, 1-3 May 1974.
<PI, UR>

[Minsky 1974b]
Minsky, N., *Protection of Data Bases and the Process of User Data-Base Interaction*, Technical Report SOSAP-TR-11, Department of Computer Science, Rutgers University, New Brunswick NJ, September 1974.
<PI>

[Minsky 1976]
Minsky, N., "Files with Semantics", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Washington D.C., 2-4 June 1976.
<DM, S>

[Mitoma 1975]
Mitoma, M. F., and K. B. Irani, "Automating Data Base Schema Design and Optimization", *Proceedings of International Conference on Very Large Data Bases*, Framingham MA, 22-24 September 1975.
<LD>

[Miyamoto 1976]
Miyamoto, I., "Some Considerations in Database Application Programming", *Proceedings of Second International Conference on Software Engineering*, San Francisco CA, 13-15 October 1976.
<A>

[Mizumoto 1977]
Mizumoto, M., M. Umano, and K. Tanaka, "Implementation of a Fuzzy-Set Theoretic Data Structure System", *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
<S>

[Montgomery 1972]
Montgomery, C. A., "Is Natural Language an Unnatural Query Language?", *Proceedings of ACM National Conference*, New York NY, 1972.
<NLI>

[Montgomery 1977]
Montgomery, C. A., "Natural Language Knowledge Processing", *Proceedings of National Computer Conference*, Dallas TX, 13-16 June 1977.
<NLI, S>

[Morgan 1970]
Morgan, H. L., "An Interrupt Based Organization for Management Information Systems", *Communications of the ACM*, Volume 13, Number 12, December 1970.
<SI>

[Moulin 1976]
Moulin, P, J. Randon, and M. Teboul, "Conceptual Models as a Data Design Tool", *Modelling in Data Base Management Systems* (editor Nijssen, G. M.), North Holland, 1976.
<DM, LD>

[MRI 1972]
MRI Systems Corporation, *System 2000 General Information Manual*, Austin TX, 1972.
<HDM, I>

[Mylopoulos 1975a]
Mylopoulos, J., S. A. Schuster, and D. Tsichritzis, "A Multi-Level Relational System", *Proceedings of National Computer Conference*, Anaheim CA, 19-22 May 1975.
<I, RDM>

[Mylopoulos 1975b]
Mylopoulos, J., P. Cohen, A. Borgida, and L. Sugar, "Semantic Networks and the Generation of Context", *Proceedings of International Joint Conference on Artifical Intelligence*, Tbilisi, Georgia, USSR, 3-8 September 1975.
<NLI, S>

[Mylopoulos 1976]
Mylopoulos, J., A. Borgida, P. Cohen, N. Roussopoulos, J. Tsotsos, and H. Wong, "TORUS: A Step Towards Bridging the Gap Between Data Bases and the Casual User", *Information Systems*, Volume 2, Number 2, Pages 49-64, 1976.
<NLI>

[Nahouraii 1976]
Nahouraii, E., L. O. Brooks, and A. F. Cerdenas, "An Approach to Data Communication Between Different Generalized Data Base Management Systems", *Systems for Large Data Bases* (editors Lockemann, P. C. and E. J. Neuhold), North Holland, 1976.
<DM>

[Nations 1978]
Nations, J. and S. Y. W. Su, "Some DML Instruction Sequences for Application Program Analysis and Conversion", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Austin TX, 31 May - 2 June 1978.
<DM, PI>

[Navathe 1978]
Navathe, S. B. and M. Schkolnick, "View Representation in Logical Database Design", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Austin TX, 31 May - 2 June 1978.
<LD, MUV>

[Nicolas 1976]
Nicolas, G. S. and J. W. Lewis, "A Mumps-Based Relational Data Base System", *Proceedings of ACM National Conference*, Houston TX, 20-22 October 1976.
<I, RDM>

[Nicolas 1978]
Nicolas, J. M., "Mutual Dependencies and Some Results on Undecomposable Relations", *Proceedings of International Conference on Very Large Data Bases*, West Berlin, West Germany, 13-15 September 1978.
<FD, RDM>

[Nijssen 1974]
Nijssen, G. M., "Data Structuring in the DDL and Relational Data Model", *Data Base Management* (editors Klimbie, J. W and K. L. Koffeman), North Holland, 1974.
<LD, NDM, RDM>

[Nijssen 1975a]
Nijssen, G. M., "Set and CODASYL Set or Coset", *Data Base Description* (editors Douque,

B. C. M. and G. M. Nijssen), North Holland, 1975.
<NDM>

[Nijssen 1975b]
Nijssen, G. M., "Two Major Flaws in the CODASYL DDL 1973 and Proposed Corrections", *Information Systems*, Volume 1, Number 4, Pages 115-132, 1975.
<NDM>

[Nijssen 1976]
Nijssen, G. M., "A Gross Architecture for the Next Generation Data Base Management Systems", *Modelling in Data Base Management Systems* (editor Nijssen, G. M.), North Holland, 1976.
<GI>

[Nijssen 1977]
Nijssen, G. M., "Architecture for the Next Generation of Data Management Systems", *Information Processing* '77, North Holland, 1977.
<GI>

[Nordstrom 1976]
Nordstrom, B., "An Outline of a Mathematical Model for the Definition and Manipulation of Data", *Proceedings of ACM SIGPLAN/SIGMOD Conference on Data: Abstraction, Definition, and Structure*, Salt Lake City UT, 22-24 March 1976.
<DM>

[Notley 1971]
Notley, M. G., *On the Design of a Graphic Display Language for Interrogating a Large Data Base*, IBM Scientific Centre Report UKSC-0004, Peterlee, England, 1974.
<NI>

[Novak 1976a]
Novak, D. O. and B. K. Kahn, *A Framework for Logical Database Design*, Working Paper 76 DE3-1, Data Translation Project, University of Michigan, Ann Arbor MI, 1976.
<LD>

[Novak 1976b]
Novak, D. O. and J. P. Fry, "The State of the Art in Database Design", *Proceedings of Fifth Texas Conference on Computing Systems*, Austin TX, 18-19 October 1976.
<LD>

[Olle 1974a]
Olle, T. W., "Current and Future Trends in Data Base Management Systems", *Information*

Processing '74, North Holland, 1974.
<GI>

[Olle 1974b]
Olle, T. W. "Data Definition Spectrum and Procedurality Spectrum in Data Base Management Systems", *Data Base Management* (editors Klimbie, J. W. and K. L. Koffeman), North Holland, 1974.
<DM, LD, NI, PI>

[Olle 1975]
Olle, T. W., "An Analysis of the Flaws in the Schema DDL and Proposed Improvements", *Data Base Description* (editors Douque, B. C. M. and G. M. Nijssen), North Holland, 1975.
<NDM>

[O'Neill 1977]
O'Neill, D. M. and S. B. Guthrey, *The Syntax and Semantics of Functional Dependency*, Technical Report, Bell Telephone Laboratories, Holmdel NJ, 1977.
<FD>

[Osman 1976]
Osman, I. M., "A Solution of Some Logical Problems of Defined Relations", *Proceedings of ACM National Conference*, Houston TX, 20-22 October 1976.
<LD, RDM>

[Owens 1971]
Owens, R. C., "Evaluation of Access Authorization Characteristics of Derived Data Sets", *Proceedings of ACM SIGFIDET Conference on Data Description, Access, and Control*, San Diego CA, 11-12 November 1971.
<IR>

[Owens 1972]
Owens, P. J., "Phase II - A Data Base Management Modelling System", *Information Processing* '72, North Holland, 1972.
<A>

[Palmer 1978a]
Palmer, I., "Practicalities in Applying a Formal Methodology to Data Analysis", *Proceedings of NYU Symposium on Database Design*, New York NY, 18-19 May 1978.
<LD, S>

[Palmer 1978b]
Palmer, I., "Record Subtype Facilities in Database Systems", *Proceedings of International*

Conference on Very Large Data Bases, West Berlin, West Germany, 13-15 September 1978.
<DM, MUV>

[Paolini 1977a]
Paolini, P. and G. Pelagatti, *Formal Definition of Data Models*, Technical Report 77-2, Institute of Electronics, Milan, Italy, 1977.
<DM>

[Paolini 1977b]
Paolini, P. and G. Pelagatti, "Formal Definition of Mappings in a Data Base", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Toronto, Canada, 3-5 August 1977.
<DM>

[Paredaens 1975]
Paredaens, J., *The Definition of Functions in a Relational Retrieval Language*, Technical Report R306, M.B.L.E. Research Laboratory, Brussels, Belgium, 1975.
<FD, NI, RDM>

[Pecherer 1975a]
Pecherer, R. M., *Efficient Evaluation of Expressions in a Relational Algebra*, Electronics Research Laboratory Report ERL-M510, University of California, Berkeley CA, 19 February 1975.
<NI, RDM>

[Pecherer 1975b]
Pecherer, R. M., *Efficient Retrieval in Relational Data Base Systems*, Electronics Research Laboratory Report ERL-M547, University of California, Berkeley CA, 2 October 1975.
<NI, RDM>

[Pecherer 1976]
Pecherer, R. M., "Efficient Exploration of Product Spaces", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Washington D. C., 2-4 June 1976.
<NI, RDM>

[Pelagatti 1977]
Pelagatti, G., P. Paolini, and G. Bracchi, "Mappings in Data Base Systems", *Information Processing* '77, North Holland, 1977.
<DM, FD>

[Petrick 1973]
Petrick, S. R., *Semantic Interpretation in the REQUEST System*, IBM Research Report

RC4457, Yorktown Heights NY, 1973.
<NLI>

[Petrick 1976]
Petrick, S. R., "On Natural Language Based Computer Systems", *IBM Journal of Research and Development*, Volume 20, Number 4, Pages 314-325, July 1976.
<NLI>

[Pirotte 1974]
Pirotte, A., and P. Wodon, *A Comprehensive Formal Query Language for a Relational Data Base: FQL*, Technical Report R283, M.B.L.E. Research Laboratory, Brussels, Belgium, 1974.
<NI, RDM>

[Pirotte 1976]
Pirotte, A., *Explicit Description of Entities and Their Manipulation in Languages for the Relational Data Model*, Technical Report, M.B.L.E. Research Laboratory, Brussels, Belgium, 1976.
<NI, RDM, S>

[Pirotte 1977]
Pirotte, A., *The Entity - Property - Association Model: An Information-Oriented Data Base Model*, Technical Report, M.B.L.E. Research Laboratory, Brussels, Belgium, 1977.
<DM, S>

[Prendergast 1972]
Prendergast, S. L., "Selecting a Data Management System", *Computer Decisions*, Volume 12, August 1972.
<GI>

[Prenner 1977]
Prenner, C. J., *A Uniform Notation for Expressing Queries*, Electronics Research Laboratory Report ERL-M7760, University of California, Berkeley CA, 8 September 1977.
<NI, PI, RDM>

[Quillian 1968]
Quillian, R., "Semantic Memory", *Semantic Information Processing* (editor Minsky, M.), MIT Press, 1968.
<NLI, S>

[Raver 1977]
Raver, N. and G. U. Hubbard, "Automated Logical Data Base Design: Concepts and

Applications", *IBM Systems Journal*, Volume 3, Pages 287-312, 1977.
<LD>

[Reisner 1975]
Resiner, P., R. F. Boyce, and D. D. Chamberlin, "Human Factors Evaluation of Two Data Base Query Languages: SQUARE and SEQUEL", *Proceedings of National Computer Conference*, Anaheim CA, 19-22 May 1975.
<NI, RDM, UC>

[Reisner 1976]
Reisner, P., *The Use of Psychological Experimentation as an Aid to Development of a Query Language*, IBM Research Report RJ1707, San Jose CA, January 1976.
<NI, RDM, UC>

[Reisner 1977]
Reisner, P., "The Use of Psychological Experimentation as an Aid to Development of a Query Language", *IEEE Transactions on Software Engineering*, Volume SE-3, Number 3, May 1977.
<NI, RDM, UC>

[Rissanen 1973]
Rissanen, J. and C. Delobel, *Decomposition of Files, A Basis for Data Storage and Retrieval*, IBM Research Report RJ1220, San Jose CA, 10 May 1973.
<DM>

[Rissanen 1977]
Rissanen, J. J., *Independent Components of Relations*, IBM Research Report RJ1899, San Jose CA, January 1977.
<RDM>

[Robinson 1967]
Robinson, J. A., "A Review of Automatic Theorem Proving", *Proceedings of Symposium in Applied Mathematics*, American Mathematical Society, Providence RI, 1967.
<DI>

[Robinson 1975a]
Robinson, K. A., "Data Base - The Ideas Behind the Ideas", *The Computer Journal*, Volume 18, Number 1, January 1975.
<GI>

[Robinson 1975b]
Robinson, K. A., "An Analysis of the Uses of the CODASYL Set Concept", *Data Base*

Description (editors Douque, B. C. M. and G. M. Nijssen), North Holland, 1975.
<LD, NDM>

[Rothnie 1972]
Rothnie, J. B., *The Design of Generalized Data Management Systems*, Ph. D. thesis, Department of Civil Engineering, Massachusetts Institute of Technology, Cambridge MA, September 1972.
<DM, LD, RDM>

[Rothnie 1976]
Rothnie, J. B. and W. T. Hardgrave, "Data Model Theory: A Beginning", *Proceedings of Fifth Texas Conference on Computing Systems*, Austin TX, 18-19 October 1976.
<DM>

[Roussopoulos 1975]
Roussopoulos, N. and J. Mylopoulos, "Using Semantic Networks for Data Base Management", *Proceedings of International Conference on Very Large Data Bases*, Framingham MA, 22-24 September 1975.
<NLI, S>

[Roussopoulos 1976]
Roussopoulos, N., *A Semantic Network Model of Data Bases*, Ph.D. Thesis, Department of Computer Science, University of Toronto, Toronto, Canada, September 1976.
<DM, S>

[Roussopoulos 1977]
Roussopoulos, N., "ADD: Algebraic Data Definition", *Proceedings of Sixth Texas Conference on Computing Systems*, Austin TX, 14-15 November 1977.
<DM, LD, NI, S, SI>

[Ruth 1976]
Ruth, G. R., *Protosystem I: An Automatic Programming System Prototype*, Technical Memorandum TM-72, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge MA, July 1976.
<NI>

[Sacerdoti 1977a]
Sacerdoti, E. D., "Language Access to Distributed Data with Error Recovery", *Proceedings of International Joint Conference on Artificial Intelligence*, Cambridge MA, 22-25 August 1977.
<DM, NI, PI>

[Sacerdoti 1977b]

Sacerdoti, E. D. (editor), *Mechanical Intelligence: Research and Applications*, Technical Report, SRI International, Menlo Park CA, December 1977.
<DI, NI, NLI, RDM, S>

[Sagalowicz 1977]
Sagalowicz, D., "IDA: An Intelligent Data Access Program", *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
<DM, NI>

[Sarin 1977]
Sarin, S. K., *Automatic Synthesis of Efficient Procedures for Database Integrity Checking*, S. M. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge MA, 1977.
<DM, SI>

[Schank 1973]
Schank, R. C., "Identification of Conceptualizations Underlying Natural Language", *Computer Models of Thought and Language* (editors Schank, R. C. and K. M. Colby), W. H. Freeman, 1973.
<NLI, S>

[Schenk 1977]
Schenk, K. L. and J. R. Pinkert, "An Algorithm for Servicing Multi-Relational Queries", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Toronto, Canada, 3-5 August 1977.
<RDM>

[Schiemann 1976]
Schiemann, A., *Set of Semantic Operators for the Entity-Relationship Model*, S. M. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge MA, June 1976.
<DM, S>

[Schmid 1975a]
Schmid, H. A. and J. R. Swenson, "On the Semantics of the Relational Data Model", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, San Jose CA, 14-16 May 1975.
<RDM, S>

[Schmid 1975b]
Schmid, H. A. and P. A. Bernstein, "A Multi-level Architecture for Relational Data Base Systems", *Proceedings of International Conference on Very Large Data Bases*, Framingham

MA, 22-24 September 1975.
<GI, RDM>

[Schmid 1976]
Schmid, H. A., *An Analysis of Some Constructs for Conceptual Models*, Technical Report, University of Stuttgart, Stuttgart, West Germany, May 1976.
<DM, LD, S>

[Schmidt 1977a]
Schmidt, J. W., "Some High-Level Language Constructs for Data of Type Relation", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Toronto, Canada, 3-5 August 1977.
<NI, PI, RDM>

[Schmidt 1977b]
Schmidt, J. W., *Type Concepts for Database Definition: An Investigation Based on Extensions to Pascal*, Technical Report, University of Hamburg, Hamburg, West Germany, 1977.
<NI, PI, RDM>

[Schmidt 1977c]
Schmidt, J. W., "Some High Level Language Constructs for Data of Type Relation", *ACM Transactions on Database Systems*, Volume 2, Number 3, Pages 247-261, September 1977.
<NI, PI, RDM>

[Schmidt 1978]
Schmidt, J. W., "Type Concepts for Database Definition", *Proceedings of International Conference on Data Bases: Improving Usability and Responsiveness*, Haifa, Israel, 2-4 August 1978.
<DM, LD>

[Schneider 1975]
Schneider, L. S., and C. R. Spath, "Quantitative Data Description", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, San Jose CA, 14-16 May 1975.
<DM, LD>

[Schneider 1976]
Schneider, L. S., "A Relational View of the Data Independent Accessing Model", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Washington D.C., 2-4 June 1976.
<DM, RDM>

[Schneiderman 1974]
Schneiderman, B. and P. Scheuermann, "Structured Data Structures", *Communications of the ACM*, Volume 17, Number 10, 1974.
<DM, GI>

[Schubert 1975]
Schubert, L. K., "Extending the Expressive Power of Semantic Networks", *Proceedings of International Joint Conference on Artifical Intelligence*, Tbilisi, Georgia, USSR, 3-8 September 1975.
<NLI, S>

[Scott 1977]
Scott, G. M., "Auditing Large Scale Data Bases", *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
<SI>

[Scragg 1976]
Scragg, G. W., "Semantic Nets as Memory Models", *Computational Semantics* (editors Charniak, E. and Y. Wilks), North Holland, 1976.
<S>

[Senko 1973]
Senko, M., E. Altman, M. Astrahan, and P. Fehder, "Data Structures and Accessing in Data Base Systems", *IBM Systems Journal*, Volume 12, Number 1, 1973.
<DM>

[Senko 1974]
Senko. M. E., *Data Description Language in the Context of a Multilevel Structured Description - DIAM II with FORAL*, IBM Research Report RC5073, Yorktown Heights NY, October 1974.
<DM, LD, NI>

[Senko 1975a]
Senko, M. E., "Specifications of Stored Data Structures and Desired Output Results in DIAM II with FORAL", *Proceedings of International Conference on Very Large Data Bases*, Framingham MA, 22-24 September 1975.
<DM, NI>

[Senko 1975b]
Senko, M. E., *An Introduction to FORAL for Users*, IBM Research Report RC5263, Yorktown Heights NY, 1975.

<DM, NI>

[Senko 1975c]
Senko, M. E., "The DDL in the Context of a Multilevel Structured Description: DIAM II with FORAL", *Data Base Description* (editors Douque, B. C. M. and G. M. Nijssen), North Holland, 1975.
<DM, NI>

[Senko 1975d]
Senko, M. E., "Information Systems: Records, Relations, Sets, Entities, and Things", *Information Systems*, Volume 1, Number 1, Pages 3-14, 1975.
<DM, LD, S>

[Senko 1976a]
Senko, M. E., "DIAM II: The Binary Infological Level and Its Database Language - FORAL", *Proceedings of ACM SIGPLAN/SIGMOD Conference on Data: Abstraction, Definition, and Structure*, Salt Lake City UT, 22-24 March 1976.
<DM, NI>

[Senko 1976b]
Senko, M. E., "DIAM II and Levels of Abstraction: The Physical Device Level: A General Model for Access Methods", *Systems for Large Data Bases* (editors Lockemann, P. C. and E. J. Neuhold), North Holland, 1976.
<DM>

[Senko 1976c]
Senko, M. E., "DIAM as a Detailed Example of the ANSI/SPARC Architecture", *Modelling in Data Base Management Systems* (editor Nijssen, G. M.), North Holland, 1976.
<DM>

[Senko 1977]
Senko, M. E., "Conceptual Schemas, Abstract Data Structures, Enterprise Descriptions", *Proceedings of ACM International Computing Symposium*, Belgium, April 1977.
<DM, S>

[Senko 1978]
Senko, M. E., "FORAL LP: Design and Implementation", *Proceedings of International Conference on Very Large Data Bases*, West Berlin, West Germany, 13-15 September 1978.
<DM, NI>

[Sharman 1975]
Sharman, G. C. H., *A New Model of Relational Data Base High Level Languages*, Technical

Report TR 12.136, IBM Hursley Park Laboratory, England, February 1975.
<NI, RDM>

[Sharman 1976]
Sharman, G. C. H., "A Constructive Definition of Third Normal Form", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Washington D.C., 2-4 June 1976.
<FD, LD, RDM>

[Sharman 1977]
Sharman, G. C. H., "Update-by-Dialogue: An Interactive Approach to Database Modification", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Toronto, Canada, 3-5 August 1977.
<NI>

[Sharman 1978]
Sharman, G. C. H. and N. Winterbottom, "The Data Dictionary Facilities of NDB", *Proceedings of International Conference on Very Large Data Bases*, West Berlin, West Germany, 13-15 September 1978.
<LD, S>

[Shu 1975]
Shu, N. C., B. C. Housel, and V. Y. Lum, "CONVERT: A High Level Translation Definition Language for Data Conversion", *Communications of the ACM*, Volume 18, Number 10, October 1975.
<DM>

[Shu 1977]
Shu, N. C., B. C. Housel, R. W. Taylor, S. P. Ghosh, and V. Y. Lum, "EXPRESS: A Data EXtraction, Processing, and REStructuring System", *ACM Transactions on Database Systems*, Volume 2, Number 2, Pages 134-174, June 1977.
<DM>

[Sibley 1973]
Sibley, E. H. and R. W. Taylor, "A Data Definition and Mapping Language", *Communications of the ACM*, Volume 16, Number 12, Pages 750-759, December 1973.
<DM, LD>

[Sibley 1974a]
Sibley, E. H., "Data Management System User Requirements", *Data Base Management Systems* (editor Jardine, D. A.), North Holland, 1974.
<UR>

[Sibley 1974b]
Sibley, E. H., "On the Equivalence of Data Based Systems", *Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control*, Ann Arbor MI, 1-3 May 1974.
<DM>

[Sibley 1977a]
Sibley, E. and L. Kerschberg, "Data Architecture and Data Model Considerations", *Proceedings of National Computer Conference*, Dallas TX, 13-16 June 1977.
<DM>

[Sibley 1977b]
Sibley, E., "Standardization and Database Systems", *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
<DM, GI, UR>

[Siklossy 1977]
Siklossy, L. and D. L. Chester, "Data Bases That Talk Back", *Proceedings of Sixth Texas Conference on Computing Systems*, Austin TX, 14-15 November 1977.
<NI, UR>

[Simmons 1970]
Simmons, Q. F., "Natural Language Question Answering Systems", *Communications of the ACM*, Volume 13, Number 1, January 1970.
<NLI>

[Simmons 1977]
Simmons, R. and D. Chester, "Inferences in Qualified Semantic Networks", *Proceedings of International Joint Conference on Artificial Intelligence*, Cambridge MA, 22-25 August 1977.
<DI, S>

[Smith 1975]
Smith, J. M. and P. Y. T. Chang, "Optimizing the Performance of a Relational Data Base Interface", *Communications of the ACM*, Volume 18, Number 10, October 1975.
<NI, UC>

[Smith 1977a]
Smith, J. M. and D. C. P. Smith, "Database Abstractions: Aggregation", *Communications of the ACM*, Volume 20, Number 6, Pages 405-413, June 1977.
<LD, RDM, S>

[Smith 1977b]
Smith, J. M. and D. C. P. Smith, "Database Abstractions: Aggregation and Generalization", *ACM Transactions on Database Systems*, Volume 2, Number 2, Pages 105-133, June 1977.
<LD, RDM, S>

[Smith 1977c]
Smith, J. M. and D. C. P. Smith, *Integrated Specifications for Abstract Systems*, Technical Report UUCS-77-112, Department of Computer Science, University of Utah, Salt Lake City UT, 1977.
<LD, RDM, S>

[Smith 1978a]
Smith, J. M. and D. C. P. Smith, "Principles of Database Design", *Proceedings of NYU Symposium on Database Design*, New York NY, 18-19 May 1978.
<DM, LD, S>

[Smith 1978b]
Smith, J. M., "A Normal Form for Abstract Syntax", *Proceedings of International Conference on Very Large Data Bases*, West Berlin, West Germany, 13-15 September 1978.
<DM, LD, S>

[Sneeringer 1976]
Sneeringer, C., "LOLIPOP: A Data Manipulation Language for Data Independence", *Proceedings of Fifth Texas Conference on Computing Systems*, Austin TX, 18-19 October 1976.
<NI, PI>

[Snuggs 1974]
Snuggs, M. E., G. J. Popek, and R. J. Peterson, "Data Base System Objectives as Design Constraints", *Proceedings of ACM National Conference*, 1974.
<DM, LD, UR>

[Software AG 1974]
Software AG, *ADABAS ADASCRIPT User's Manual*, Reston VA, 1974.
<HDM, I, NDM>

[Solvberg 1977]
Solvberg, A., *A Model for Specification of Phenomena, Properties, and Information Systems*, IBM Research Report RJ2027, San Jose CA, 18 July 1977.
<DM, NI, S>

[Sorenson 1977]
Sorenson, P. G. and J. A. Wald, "PICASSO: An Aid to an End-User Facility", *Proceedings*

of ACM SIGMOD International Conference on the Management of Data, Toronto, Canada, 3-5 August 1977.
<NI>

[Sowa 1976]
Sowa, J. F., "Conceptual Graphs for a Data Base Interface", *IBM Journal of Research and Development*, Volume 20, Number 4, July 1976.
<DM, NI, S>

[Spath 1977]
Spath, C. R. and L. S. Schneider, "A Generalized End-User Facility Architecture for Relational Database Systems", *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
<NI, RDM>

[Sprowls 1976]
Sprowls, R. C., *Management Data Bases*, Wiley/Hamilton, 1976.
<A, DM>

[Steel 1975]
Steel, T. B., "Data Base Standardization: A Status Report", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, San Jose CA, 14-16 May 1975.
<GI>

[Stemple 1976]
Stemple, D. W., "A Database Management Facility for Automatic Generation of Database Managers", *ACM Transactions on Database Systems*, Volume 1, Number 1, Pages 79-94, March 1976.
<NDM>

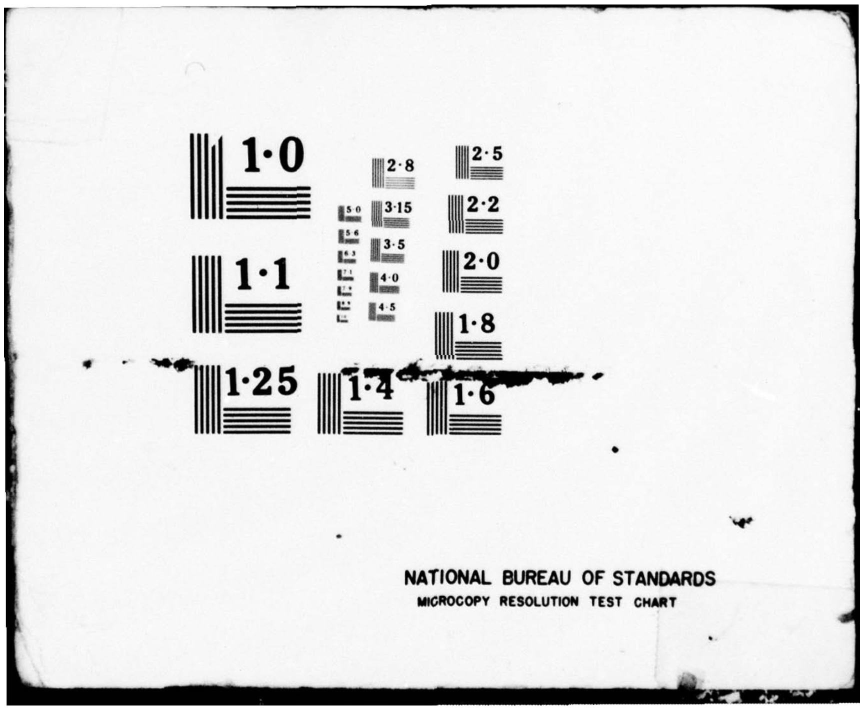
[Steuert 1974]
Steuert, J. and J. Goldman, "The Relational Data Management System: A Perspective", *Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control*, Ann Arbor MI, 1-3 May 1974.
<I, NI, PI, RDM>

[Stickney 1977]
Stickney, M. E., "The Theory of Questionnaires with Applications to Data Management", *Proceedings of COMPSAC '77*, Chicago IL, 8-11 November 1977.
<NI, UC>

AD-A068 112 MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTE--ETC F/G 9/2
A SEMANTIC DATA BASE MODEL AND ITS ASSOCIATED STRUCTURED USER I--ETC(U)
AUG 78 D MCLEOD N00014-76-C-0944
UNCLASSIFIED MIT/LCS/TR-214 NL

4 OF 4
ADA
068112

END
DATE
FILMED
6-79
DDC


1·0
2·8
2·5
3·15
2·2
5·0
5·6
3·5
2·0
1·1
4·0
4·5
1·8
1·25
1·4
1·6
NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

[Stonebraker 1974a]
Stonebraker, M. R., "A Functional View of Data Independence", *Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control*, Ann Arbor MI, 1-3 May 1974.
<LD, RDM>

[Stonebraker 1974b]
Stonebraker, M. R., *High Level Integrity Assurance in Relational Data Base Management Systems*, Electronics Research Laboratory Report ERL-M473, University of California, Berkeley CA, 16 August 1974.
<RDM, SI>

[Stonebraker 1974c]
Stonebraker, M. and E. Wong, *Access Control in a Relational Data Base Management System by Query Modification*, Electronics Research Laboratory Report ERL-M438, University of California, Berkeley CA, 14 May 1974.
<RDM>

[Stonebraker 1974d]
Stonebraker, M. and E. Wong, "Access Control in a Relational Data Base Management System", *Proceedings of ACM National Conference*, San Diego CA, November 1974.
<RDM>

[Stonebraker 1975a]
Stonebraker, M. R., *Getting Started in INGRES - A Tutorial*, Electronics Research Laboratory Report ERL-M518, University of California, Berkeley CA, 23 April 1975.
<NI, RDM>

[Stonebraker 1975b]
Stonebraker, M. "Implementation of Integrity Constraints and Views by Query Modification", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, San Jose CA, 14-16 May 1975.
<MUV, RDM, SI>

[Stonebraker 1976a]
Stonebraker, M., E. Wong, P. Kreps, and G. Held, "The Design and Implementation of INGRES", *ACM Transactions on Database Systems*, Volume 1, Number 3, Pages 189-222, September 1976.
<I, RDM>

[Stonebraker 1976b]
Stonebraker, M. R. and P. Rubenstein, "The INGRES Protection System", *Proceedings of*

ACM National Conference, Houston TX, 20-22 October 1976.
<RDM>

[Stonebraker 1977a]
Stonebraker, M., "Database Languages", *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
<NI, PI>

[Stonebraker 1977b]
Stonebraker, M. R. and L. A. Rowe, *Observations on Data Manipulation Languages and Their Embedding in General Purpose Programming Languages*, Electronics Research Laboratory Report ERL-M7753, University of California, Berkeley CA, 29 July 1977.
<NI, PI, RDM>

[Strnad 1971]
Strnad, A. L., "The Relational Approach to the Management of Data Bases", *Information Processing '71*, North Holland, 1971.
<RDM>

[Su 1976]
Su, S. Y. W., "Application Program Conversion Due to Data Base Changes", *Systems for Large Data Bases* (editors Lockemann, P. C. and E. J. Neuhold), North Holland, 1976.
<PI>

[Su 1977]
Su, S. Y. W. and B. J. Liu, "A Methodology of Application Program Analysis and Conversion Based on Database Semantics", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Toronto, Canada, 3-5 August 1977.
<PI, S>

[Summers 1975a]
Summers, R. C., C. D. Coleman, and E. B. Fernandez, "A Programming Language Extension for Access to a Shared Data Base", *Proceedings of ACM Pacific Conference*, San Francisco CA, 17-18 April 1975.
<PI>

[Summers 1975b]
Summers, R. C. and E. B. Fernandez, *Data Description for a Shared Data Base: Views, Integrity, and Authorization*, Technical Report G320-2671, IBM Scientific Center, Los Angeles CA, August 1975.
<MUV, SI>

[Sundgren 1974]
Sundgren, B., "A Conceptual Foundation for the Infological Approach to Data Bases", *Data Base Management* (editors Klimbie, J. W. and K. L. Koffeman), North Holland, 1974.
<DM, LD, S>

[Symonds 1970]
Symonds, A. J. and R. A. Lorie, "A Schema for Describing a Relational Data Base", *Proceedings of ACM SIGFIDET Workshop on Data Description and Access*, November 1970.
<LD, RDM>

[Szolovits 1977]
Szolovits, P., L. B. Hawkinson, and W. A. Martin, "An Overview of OWL, A Language for Knowledge Representation", *Natural Language for Interaction with Data Bases* (editors Rahmstorf, G. and M. Ferguson), 1977.
<NLI, S>

[Tanaka 1977]
Tanaka, Y. and T. Tsuda, "Decomposition and Composition of a Relational Database" *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
<FD, LD, RDM>

[Taylor 1974a]
Taylor, B. J. and S. C. Lloyd, "DUCHESS - A High Level Information System", *Proceedings of National Computer Conference*, Chicago IL, 6-10 May 1974.
<I>

[Taylor 1974b]
Taylor, R. W., "Data Administration and the DBTG Report", *Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control*, Ann Arbor MI, 1-3 May 1974.
<LD, NDM>

[Taylor 1975]
Taylor, R. W., "Observations on the Attributes of Database Sets", *Data Base Description* (editors Douque, B. C. M. and G. M. Nijssen), North Holland, 1975.
<NDM>

[Taylor 1976]
Taylor, R. W. and R. L. Frank, "CODASYL Data-Base Management Systems", *Computing Surveys*, Volume 8, Number 1, March 1976.
<NDM>

[Teorey 1978]
Teorey, T. J. and J. P. Fry, ***Logical Database Design: A Pragmatic Approach***, Technical Report 78DE12, Database Systems Research Group, Graduate School of Business Administration, University of Michigan, Ann Arbor MI, January 1978.
<LD>

[terBekke 1976]
ter Bekke, J. H., "A Data Manipulation Language for Relational Data Structures", ***Systems for Large Data Bases*** (editors Lockemann, P. C. and E. J. Neuhold), North Holland, 1976.
<NI, RDM>

[Thomas 1975]
Thomas, J., and J. D. Gould, "A Psychological Study of Query by Example", ***Proceedings of National Computer Conference***, Anaheim CA, 19-22 May 1975.
<NI, RDM, UC>

[Thomas 1977]
Thomas, J., "Psychological Issues in Data Base Management", ***Proceedings of International Conference on Very Large Data Bases***, Tokyo, Japan, 6-8 October 1977.
<NI, PI, UC, UR>

[Titman 1974]
Titman, P. "An Experimental Data Base System Using Binary Relations", ***Data Base Management*** (editors Klimbie, J. W. and K. L. Koffeman), North Holland, 1974.
<I, RDM>

[Todd 1976]
Todd, S. J. P., "The Peterlee Relational Test Vehicle - A System Overview", ***IBM Systems Journal***, Volume 15, Number 4, 1976.
<I, RDM>

[Todd 1977a]
Todd, S., "Automatic Constraint Maintenance and Updating Defined Relations", ***Information Processing '77***, North Holland, 1977.
<RDM, SI>

[Todd 1977b]
Todd, S. J. P., ***Relational Database Research at the IBM United Kingdom Scientific Centre***, IBM Scientific Centre Report UKSC-0093, Peterlee, England, December 1977.
<GI, RDM>

[Trigoboff 1977]
Trigoboff, M. and C. Kulikowski, "IRIS: A System for the Propogation of Inferences in a Semantic Net", *Proceedings of International Joint Conference on Artificial Intelligence*, Cambridge MA, 22-25 August 1977.
<DI, S>

[Tsichritzis 1975]
Tsichritzis, D., *Features of a Conceptual Schema*, Technical Report CSRG-56, Computer Systems Research Group, University of Toronto, Toronto, Canada, July 1975.
<DM>

[Tsichritzis 1976a]
Tsichritzis, D., "LSL: A Link and Selector Language", *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Washington D.C., 2-4 June 1976.
<DM, NI>

[Tsichritzis 1976b]
Tsichritzis, D. C. and F. H. Lochovsky, "Hierarchical Data-Base Management: A Survey", *Computing Surveys*, Volume 8, Number 1, March 1976.
<HDM>

[Tsichritzis 1976c]
Tsichritzis, D. and A. Klug, *An Example of ANSI/SPARC Architecture*, Technical Report, Computer Systems Research Group, University of Toronto, Toronto, Canada, November 1976.
<DM>

[Tsichritzis 1976d]
Tsichritzis, D. C., *Research Directions in Data Base Management Systems*, Technical Report, Computer Systems Research Group, University of Toronto, Toronto, Canada, 1976.
<GI>

[Tsichritzis 1977a]
Tsichritzis, D. C. and F. H. Lochovsky, *Data Base Management Systems*, Academic Press, 1977.
<GI, HDM, I, NDM, NI, PI, RDM>

[Tsichritzis 1977b]
Tsichritzis, D. C., *A Panache of DBMS Ideas*, Technical Report CSRG-78, Computer Systems Research Group, University of Toronto, Toronto, Canada, 1977.
<GI>

[Tyler 1977]
Tyler, J. M., C. M. Smith, R. W. Drake, M. E. Black, S. S. Cohen, T. J. Harris, J. B. McLean, K. Sperka, and W. Rehling, "A Data Base Language for FORTRAN", *Proceedings of National Computer Conference*, Seattle WA, 17-19 October 1977.
<NDM, PI>

[Uhrowczik 1973]
Uhrowczik, P. P., "Data Dictionary/Directories", *IBM Systems Journal*, Number 4, 1973.
<LD, S>

[Valle 1975]
Valle, G., *Interactive Handling of Data Base Relations: Experiments with the Relational Approach*, Technical Report, University of Bologna, Bologna, Italy, March 1975.
<NI, RDM>

[Vanduck 1977]
Vanduck, E., "Towards a More Familiar Relational Retrieval Language", *Information Systems*, Volume 2, Pages 159-169, 1977.
<NI, RDM>

[Vassiliou 1976]
Vassiliou, Y. and D. Tsichritzis, *A Front End Data Base System*, Technical Report, Department of Computer Science, University of Toronto, Toronto, Canada, 1976.
<I>

[Vetter 1977]
Vetter, M., "Database Design by Applied Data Synthesis", *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
<LD>

[Von Gohren 1973]
Von Gohren, G. L., "User Experience with Integrated Data Store (IDS)", *Data Base Management Systems* (editor Jardine, D. A.), North Holland, 1974.
<NDM, UC>

[Waghorn 1975]
Waghorn, W. J., "The DDL as an Industry Standard?", *Data Base Description* (editors Douque, B. C. M. and G. M. Nijssen), North Holland, 1975.
<NDM>

[Waltz 1977]
Waltz, D. and B. Goodman, "Writing a Natural Language Data Base System", *Proceedings*

of International Joint Conference on Artificial Intelligence, Cambridge MA, 22-25 August 1977.
<NLI>

[Waltz 1978]
Waltz, D. L., "An English Language Question Answering System", *Communications of the ACM*, Volume 21, Number 7, Pages 526-539, July 1978.
<NLI>

[Wang 1975]
Wang, C. P. and H. H. Wedekind, "Segment Synthesis in Logical Data Base Design", *IBM Journal of Research and Development*, Volume 19, Number 1, January 1975.
<LD>

[Weber 1976a]
Weber, H., "A Semantic Model of Integrity Constraints on a Relational Data Base", *Modelling in Data Base Management Systems* (editor Nijssen, G. M.), North Holland, 1976.
<SI>

[Weber 1976b]
Weber, H., *The D-Graph Model of Large Shared Data Bases: A Representation of Integrity Constraints and Views as Abstract Data Types*, IBM Research Report RJ1875, San Jose CA, 1976.
<DM, MUV, S, SI>

[Weiner 1977]
Weiner, J. L., "Deriving Data Base Specifications from User Queries", *Proceedings of the Second Berkeley Workshop on Distributed Data Management and Computer Networks*, Berkeley CA, 25-27 May 1977.
<LD, NI>

[Weldon 1976]
Weldon, J. L. and T. Navathe, "An Attribute-Based File Organization for a Relational Data Base", *Proceedings of ACM National Conference*, Houston TX, 20-22 October 1976.
<RDM>

[Wellis 1972]
Wellis, M. E., W. Katke, J. Olson, and S. C. Yang, "SIMS - An Integrated User-Oriented Information System", *Proceedings of Fall Joint Computer Conference*, 1972.
<A, UC>

[Wells 1976]
Wells, M. B. and F. L. Cornwall, "A Data Type Encapsulation Scheme Utilizing Base Language Operators", *Proceedings of ACM SIGPLAN/SIGMOD Conference on Data: Abstraction, Definition, and Structure*, Salt Lake City UT, 22-24 March 1976.
<PI>

[Westgaard 1975]
Westgaard, R. E., "A COBOL Data Base Facility for the Relational Data Model", *Proceedings of ACM Pacific Conference*, San Francisco CA, 17-18 April 1975.
<PI, RDM>

[Whitney 1972]
Whitney, V. K. M., "RDMS: A Relational Data Management System", *Proceedings of Fourth International Symposium on Computer and Information Sciences*, Miami Beach FL, 14-16 December 1972.
<I, RDM>

[Whitney 1974]
Whitney, V. K. M., "Relational Data Management Implementation Techniques", *Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control*, Ann Arbor MI, 1-3 May 1974.
<RDM>

[Wiederhold 1977]
Wiederhold, G., *Database Design*, McGraw Hill, 1977.
<A, DM, LD, NI, PI, S>

[Wilkes 1972]
Wilkes, M. V., "On Preserving the Integrity of Data Bases", *The Computer Journal*, Volume 15, Number 3, 1972.
<SI>

[Winograd 1975a]
Winograd, T., *Understanding Natural Language*, Academic Press, 1975.
<NLI>

[Winograd 1975b]
Winograd, T., "Frame Representations and the Declarative - Procedural Controversy", *Representation and Understanding* (editors Bobrow, D. G. and A. Collins), Academic Press, 1975.
<NLI, S>

[Wong 1971]
Wong, E. and T. C. Chiang, "Canonical Structure in Attribute Based File Organizations", *Communications of the ACM*, Volume 14, Pages 593-597, 1971.
<DM>

[Wong 1976]
Wong, E. and K. Youssefi, "Decomposition - A Strategy for Query Processing", *ACM Transactions on Database Systems*, Volume 1, Number 3, Pages 223-241, September 1976.
<NI, RDM>

[Wong 1977a]
Wong, K. C., "Interval Hierarchies and Their Application to Predicate Files", *ACM Transactions on Database Systems*, Volume 2, Number 3, Pages 223-232, September 1977.
<NI>

[Wong 1977b]
Wong, K. K. T., "Two Views of Data Semantics: A Survey of Data Models in Artificial Intelligence and Database Management", *Infor*, Volume 15, Number 3, Pages 344-382, October 1977.
<DM, NI, S, SI>

[Woods 1975]
Woods, W. A., "What's in a Link: Foundations for Semantic Networks", *Representation and Understanding* (editors Bobrow, D. G. and A. Collins), Academic Press, 1975.
<S>

[Yao 1978]
Yao, B., "An Integrated Approach to Logical Database Design", *Proceedings of NYU Symposium on Database Design*, New York NY, 18-19 May 1978.
<LD>

[Yasaki 1977]
Yasaki, E. K., "The Many Faces of the DBA", *Datamation*, Volume 23, Number 5, Pages 75-79, May 1977.
<GI>

[Yeh 1977]
Yeh, R., "Application of Software Engineering Methodology to the Design of Database Systems", *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
<LD>

[Zadeh 1965]
Zadeh, L. A., "Fuzzy Sets", *Information Control*, Volume 8, Page 338, 1965.
<DM, S>

[Zadeh 1973]
Zadeh, L. A., *The Concept of a Linguistic Variable and Its Application to Approximate Reasoning*, Electronics Research Laboratory Report ERL-M411, University of California, Berkeley CA, 1973.
<DM, PI>

[Zadeh 1975]
Zadeh, L. A., *Calculus of Fuzzy Restrictions*, Electronics Research Laboratory Report ERL-M502, University of California, Berkeley CA, 19 February 1975.
<DM, S>

[Zaniolo 1976]
Zaniolo, C., *Analysis and Design of Relational Schemata for Database Systems*, Technical Report UCLA-ENG-7669, University of California, Los Angeles, Los Angeles CA,, July 1976.
<LD, RDM>

[Zaniolo 1977]
Zaniolo, C. A., "Relational Views in a Data Base System: Support for Queries", *Proceedings of COMPSAC '77*, Chicago IL, 8-11 November 1977.
<MUV, NI, RDM>

[Zloof 1974]
Zloof, M. M., *Query by Example*, IBM Research Report RC4917, Yorktown Heights NY, 2 July 1974.
<NI, RDM>

[Zloof 1975a]
Zloof, M. M., "Query by Example", *Proceedings of National Computer Conference*, Anaheim CA, 19-22 May 1975.
<NI, RDM>

[Zloof 1975b]
Zloof, M. M., "Query by Example: The Invocation and Definition of Tables and Forms", *Proceedings of International Conference on Very Large Data Bases*, Framingham MA, 22-24 September 1975.
<LD, NI, RDM>

[Zloof 1975c]
Zloof, M. M., *Query by Example: Operations on Hierarchical Data Bases*, IBM Research Report RC5491, Yorktown Heights NY, 26 June 1975.
<NI, RDM>

[Zloof 1975d]
Zloof, M. M., *Query by Example: Operations on the Transitive Closure*, IBM Research Report RC5526, Yorktown Heights NY, 17 July 1975.
<NI, RDM>

[Zloof 1975e]
Zloof, M. M. and S. P. deJong, *The System for Business Automation (SBA): Programming Language*, IBM Research Report RC5302, Yorktown Heights NY, 10 March 1975.
<A, NI>

[Zloof 1977a]
Zloof, M. M. and S. P. deJong, "The System for Business Automation (SBA): Programming Language", *Communicatons of the ACM*, Volume 20, Number 6, Pages 385-396, June 1977.
<A, NI, RDM>

[Zloof 1977b]
Zloof, M. M., "Query-by-Example: A Data Base Query Language", *IBM Systems Journal*, Volume 16, Number 4, Pages 324-343, 1977.
<NI, RDM>

[Zloof 1978]
Zloof, M. M., "Design Aspects of the Query-by-Example Data Base Management Language", *Proceedings of International Conference on Data Bases: Improving Usability and Responsiveness*, Haifa, Israel, 2-4 August 1978.
<LD, NI, RDM>

[Zook 1976]
Zook, W., K. Youssefi, N. Whyte, P. Rubenstein, P. Kreps, G. Held, J. Ford, R. Berman, and E. Allman, *INGRES Reference Manual - Version 6*, Electronics Research Laboratory Report ERL-M579, University of California, Berkeley CA, 6 April 1976.
<I, RDM>

Note

The following is a list of correspondences between conference proceedings and books in which the proceedings were published:

Data Base Description (editors Douque, B. C. M. and G. M. Nijssen), North Holland, 1975 is *Proceedings of IFIP TC-2 Special Working Conference on the DDL*, Namur, Belgium, 13-17 January 1975.

Data Base Systems (editor Rustin, R.), Prentice Hall, 1971 is *Proceedings of Courant Computer Science Symposia 6*, New York NY, 24-25 May 1971.

Data Base Management (editors Klimbie, J. W. and K. L. Koffeman), North Holland, 1974 is *Proceedings of IFIP TC-2 Working Conference on Data Base Management Systems*, Cargese, Corsica, 1-5 April 1974.

Modelling in Data Base Management Systems (editor Nijssen, G. M.), North Holland, 1976 is *Proceedings of IFIP TC-2 Working Conference on Modelling in Data Base Management Systems*, Freudenstadt, Black Forest, West Germany, 5-9 January 1976.

Pattern Recognition and Artificial Intelligence (editor Chen, C. H.), Academic Press, 1976 is *Proceedings of the Joint Workshop on Pattern Recognition and Artificial Intelligence*, Hyannis MA, 1-3 June 1976.

Information Processing '68, Information Processing '71, Information Processing '74, and *Information Processing '77* are *Proceedings of IFIP Congress*.

Systems for Large Data Bases (editors Lockemann, P. C. and E. J. Neuhold), North Holland, 1976, is *Proceedings of International Conference on Very Large Data Bases*, Brussels, Belgium, 8-10 September 1976.

Figure 2-1. Example Operations on a Relational Data Base

For creating and deleting relations:

create_domain
delete_domain
create_relation
delete_ relation

For creating new relations (for retrieval or addition to a data base):

restrict
project
join
intersect
union
difference
copy_relation

For changing data in relations:

insert_tuple
delete_tuple
update_tuple

For changing the structure of relations:

add_column_to_relation
delete_column_from_relation

Figure 2-2a. A Relational Schema for the TMAE

(Note: Underlying domains are omitted for simplicity.)

SHIP (Ship_name, Ship_type, Hull_number, Call_sign, Is_doctor_on_board, Country, Engagement, Maximum_speed, Fuel_type)

ASSIGNMENT (Captain, Ship_name, Date_start, Date_end)

CARGO (Ship_name, Quantity)

PORT (Port_name, Location, Country)

PORT_OF_SHIP (Name, Ship_name)

BANNED_SHIP (Ship_name, Date_effective)

POSITION_REPORT (Ship_name, Convoy_name, Date, Time, Is_ship_in_port, Port_name, Geo_coordinate)

REPORT_PRECISION (Ship_name, Convoy_name, Date, Time, Precision_type, Circular_radius, Ellipse_major_axis, Ellipse_minor_axis, Ellipse_inclination)

SAILING_PLAN (Ship_name, Date_filed, Involved_convoy)

SAILING_PLAN_STOPS (Ship_name, Convoy_name, Date_filed, Port_name, Date_arriving)

Figure 2-2a. (continued) A Relational Schema for the TMAE

COMMERCIAL_SHIP (Ship_name, Weight)

MILITARY_SHIP (Ship_name, Displacement)

OIL_TANKER (Ship_name, Date_of_last_inspection, Hull_type)

INCIDENT (Incident_number, Date, Time, Description)

INCIDENT_SHIPS (Incident_number, Ship_name)

INSPECTION (Ship_name, Inspector, Date, Violations, Disposition)

INSPECTOR (Inspector_name)

CONVOY (Convoy_name, Ship_name)

SHIP_TYPE (Ship_type, Cargo_type)

Figure 2-2b. A List of Underlying Domains for the TMAE Relational Data Base

SHIP_NAME
SHIP_TYPE
SHIP_HULL_NUMBER
RADIO_CALL_SIGN
YES_NO
COUNTRY_NAME
ENGAGEMENT
KNOTS
FUEL_TYPE
CAPTAIN_NAME
DATE
CARGO_TYPE
TONS
PORT_NAME
GEO_COORDINATE
CONVOY_NAME
TIME
PRECISION_TYPE
MILES
DEGREES
HULL_TYPE
INCIDENT_NUMBER
INCIDENT_DESCRIPTION
PERSON_NAME
INSPECTION_VIOLATION
INSPECTION_DISPOSITION

Figure 2-3. Views for the TMAE

CURRENT_SHIP_CAPTAIN (Ship_name, Captain)

SHIP_WITH_FLAG (Ship_name, Ship_type, Hull_number, Call_sign, Is_doctor_on_board, Flag, Engagement, Maximum_speed, Fuel_type)

CURRENT_SHIP_POSITION (Ship_name, Geo_coordinate, Port_name)

OIL_TANKER_MISHAP (Incident_number, Date, Time, Description, Involved_ship)

SHIP_DUE_FOR_INSPECTION (Ship_name, Ship_type, Hull_number, Call_sign, Is_doctor_on_board, Country, Engagement, Maximum_speed, Fuel_type)

CONVOY_SIZE (Convoy_name, Size)

SHIP_AT_SEA (Ship_name)

DANGEROUS_SHIP (Ship_name, Ship_type)

Figure 3-1. An SDM Schema for the TMAE

```
SHIPS
  description: all ships with potentially hazardous cargoes that
    may enter U.S. coastal waters
  kind: concrete object
  member attributes:
    Name
      value class: SHIP_NAMES
    Hull_number
      value class: HULL_NUMBERS
    Type
      description: the kind of ship, e.g., merchant or fishing
      value class: SHIP_TYPE_NAMES
    Country_of_registry
      value class: COUNTRIES
    Name_of_home_port
      value class: PORT_NAMES
    Cargo_types
      description: the type(s) of cargo the ship can carry
      value class: CARGO_TYPE_NAMES
      multi-valued
    Captain
      description: the current captain of the ship
      derivation: Officer of match to ASSIGNMENTS on Ship
    Engines
      kind: component
      value class: ENGINES
      multi-valued
    Oil-burning_engines
      derivation: restrict Engines where Kind_of_engine =
        'oil burning'
      kind: component
      multi-valued
  identifiers:
    Name
    Hull_number
```

Figure 3-1 (continued). An SDM Schema for the TMAE

```
INSPECTIONS
    description:  inspections of oil tankers
    kind:  point event
    member attributes:
        Tanker
            description:  the tanker inspected
            kind:  participant
            value class:  OIL_TANKERS
        Date
            value class:  DATES
        Order_for_tanker
            description:  the ordering of the inspections for a tanker
                with the most recent inspection having value 1
            derivation:  order by decreasing Date within Tanker
    class attributes:
        Number
            description:  the number of oil tankers in the data base
            derivation:  number of members in this class
    identifiers:
        Tanker + Date

COUNTRIES
    kind:  concrete object
    member attributes:
        Name
            value class:  COUNTRY_NAMES
        Ships_registered_here
            derivation:  invert Country_of_registry of SHIPS
            multi-valued
    identifiers:
        Name
```

Figure 3-1 (continued). An SDM Schema for the TMAE

```
OFFICERS
    description:  all certified officers of ships
    kind:  concrete object
    member attributes:
        Name
            value class:  PERSON_NAMES
        Country_of_license
            value class:  COUNTRIES
        Date_commissioned
            value class:  DATES
        Commander
            description:  the officer in direct command of this officer
            value class:  OFFICERS
        Superiors
            derivation:  merge members in repeat over Commander
            multi-valued
        Subordinates
            derivation:  invert Superiors of OFFICERS
            multi-valued
    identifiers:
        Name

ENGINES
    kind:  concrete object
    member attributes:
        Serial_number
            value class:  ENGINE_SERIAL_NUMBERS
        Kind_of_engine
            value class:  ENGINE_TYPE_NAMES
    identifiers:
        Serial_number
```

Figure 3-1 (continued). An SDM Schema for the TMAE

```
INCIDENTS
    description: ship accidents
    kind: point event
    member attributes:
        Involved_ship
            kind: participant
            value class: SHIPS
        Date
            value class: DATES
        Description
            description: textual explanation of the accident
            value class: INCIDENT_DESCRIPTIONS
        Involved_captain
            kind: concrete object
            value class: OFFICERS
    identifiers:
        Involved_ship + Date + Description

ASSIGNMENTS
    description: assignments of captains to ships
    kind: duration event
    member attributes:
        Officer
            kind: participant
            value class: OFFICERS
        Ship
            kind: participant
            value class: SHIPS
    identifiers:
        Officer + Ship
```

Figure 3-1 (continued). An SDM Schema for the TMAE

```
OIL_TANKERS
    derivation: restrict SHIPS where Cargo_types contains 'oil'
    member attributes:
        Hull_type
            description: specification of single or double hull
            value class: HULL_TYPE_NAMES
        Is_tanker_banned?
            derivation: if exists in BANNED_SHIPS
        Inspections
            derivation: invert Tanker of INSPECTIONS
            multi-valued
        Number_of_times_inspected
            derivation: number of unique members in Inspections
        Last_inspection
            derivation: invert Tanker of MOST_RECENT_INSPECTIONS
        Last_two_inspections
            derivation: restrict Inspections where
                Order_for_tanker <= 2
        Date_last_examined
            derivation: same as Last_inspection.Date
    class-determined attributes:
        Absolute_top_speed
            value class: KNOTS
        Top_speed_in_miles_per_hour
            derivation: = Absolute_top_speed / 1.1

LIBERIAN_OIL_TANKERS
    derivation: restrict OIL_TANKERS where Country.Name = 'Liberia'
    member attributes:
        Oil_spills_involved_in
            derivation: invert Involved_ship of OIL_SPILLS

MERCHANT_SHIPS
    derivation: restrict SHIPS where Type = 'merchant'
```

Figure 3-1 (continued). An SDM Schema for the TMAE

```
OIL_SPILLS
    derivation: restrict INCIDENTS where Description = 'oil spill'
    member attributes:
        Amount_spilled
            value class: GALLONS
        Severity
            derivation: = Amount_spilled / 100000

MOST_RECENT_INSPECTIONS
    derivation: restrict INSPECTIONS where Order_for_tanker = 1

DANGEROUS_CAPTAINS
    description: captains who have been involved in an accident
    derivation: restrict OFFICERS where is a value of
        Involved_captain of INCIDENTS

BANNED_SHIPS
    description: ships banned from U.S. coastal waters
    derivation: subset of SHIPS
    member attributes:
        Date_banned
            value class: DATES

OIL_TANKERS_REQUIRING_INSPECTION
    derivation: subset of OIL_TANKERS

BANNED_OIL_TANKERS
    derivation: extract common members in BANNED_SHIPS,
        OIL_TANKERS

SAFE_SHIPS
    description: ships which are considered good risks
    derivation: extract missing members in SHIPS but not in
        BANNED_SHIPS

SHIPS_TO_BE_MONITORED
    description: ships which are considered bad risks
    derivation: merge members in BANNED_SHIPS and
        OIL_TANKERS_REQUIRING_INSPECTION
```

Figure 3-1 (continued). An SDM Schema for the TMAE

```
SHIP_TYPES
    description: abstract types of ships
    derivation: abstract SHIPS on common value of Type
        defined instances classes are MERCHANT_SHIPS
    member attributes:
        Number_of_instances
            description: the number of instances of the
                type of ship
            derivation: number of members in Instances
        Number_of_ships_of_this_type
            derivation: same as Number_of_instances

CONVOYS
    derivation: primitive aggregate of SHIPS
    member attributes:
        Oil_tanker_constituents
            description: the oil tankers that are in the convoy
                (if any)
            derivation: restrict Constituents where
                Cargo_types contains 'oil'
            multi-valued
```

Figure 3-1 (continued). An SDM Schema for the TMAE

CARGO_TYPE_NAMES
  description: the types of cargo
  derivation: restrict STRINGS

COUNTRY_NAMES
  derivation: subset of STRINGS

DATES
  description: calendar dates in the form "8/21/78"
  derivation: restrict STRINGS

ENGINE_SERIAL_NUMBERS
  description: ten digit numbers
  derivation: restrict NUMBERS

ENGINE_TYPE_NAMES
  derivation: restrict STRINGS

GALLONS
  derivation: restrict NUMBERS

HULL_NUMBERS
  description: ten to fifteen digit numbers
  derivation: restrict NUMBERS

Figure 3-1 (continued). An SDM Schema for the TMAE

HULL_TYPE_NAMES
    description: single or double
    derivation: subset of STRINGS

INCIDENT_DESCRIPTIONS
    description: textual description of the accident
    derivation: restrict STRINGS

KNOTS
    derivation: restrict NUMBERS

PORT_NAMES
    derivation: restrict STRINGS

PERSON_NAMES
    derivation: restrict STRINGS

SHIP_NAMES
    derivation: restrict STRINGS

SHIP_TYPE_NAMES
    description: the names of the ship types, e.g., merchant
    derivation: restrict STRINGS

Figure 3-2. SDM Entity Types

1. object
    a. concrete object
    b. abstract object
        i. abstraction
        ii. aggregate
2. event
    a. point event
    b. duration event
3. name

Figure 3-3. Example Classes and Interclass Connections

```
              SHIP_TYPES                    CONVOYS
                  ^                            ^
                  |                            |
               abstract                    aggregate
      (on common value of Type)           (primitive)
                  |                            |
                  -----------      ----------
                            |      |
                  base concrete object class
                             SHIPS
                             | | |
      ------------------- | -------------
      |                   |              |
   restrict               |            subset
(where Type = 'merchant') |              |
      |                   |              v
      v                   |         BANNED_SHIPS
 MERCHANT_SHIPS           |
                          |
                       restrict
           (where Cargo_types contains 'oil')
                          |
                          v
                      OIL_TANKERS
```

Figure 3-4a. An Example of the Interclass Connection Merge Members

```
                  base concrete object class
                           SHIPS
                            | |
               --------------- ---------------
               |                             |
            restrict                       subset
   (where Cargo_types contains 'oil')        |
               |                        BANNED_SHIPS
               v                             |
          OIL_TANKERS                        |
               |                             |
            subset                           |
               |                             |
               v                             |
 OIL_TANKERS_REQUIRING_INSPECTION            |
               |                             |
               --------------- ---------------
                            | |
                            \ /
                             |
                       merge members
                             |
                             v
                   SHIPS_TO_BE_MONITORED
```

Figure 3-4b. An Example of Extract Common Members

```
                 base concrete object class
                           SHIPS
                            | |
           ---------------- ----------------
           |                               |
        restrict                         subset
(where Cargo_types contains 'oil')         |
           |                          BANNED_SHIPS
           v                               |
      OIL_TANKERS                          |
           |                               |
           ---------------- ----------------
                            | |
                            \ /
                             |
                  extract common members
                             |
                             v
                    BANNED_OIL_TANKERS
```

Figure 3-4c. An Example of Extract Missing Members

```
base concrete object class
        SHIPS
         | |
   ------ ------
   |           |
   |         subset
   |           |
   |           v
   |       BANNED_SHIPS
   |           |
   |           |
   ------ ------
         | |
         \ /
          |
extract missing members
          |
          v
      SAFE_SHIPS
```

Figure 3-5. Interclass Connection Types

Connection Type	Limitation
restrict	
subset	
merge members	there is a common root class
extract common members	there is a common root class
extract missing members	there is a common root class
abstract	the grouping expression is not circular
aggregate	

Figure 3-6. Example Name Class Definitions

```
ENGINE_SERIAL_NUMBERS
    derivation: restrict NUMBERS
        where format is
            "H"
            number where integer and >= 1 and <= 999
            "-"
            number where integer and >= 0 and <= 999999

DATES
    description: calendar dates in the form "8/21/78"
    derivation: restrict STRINGS
        where format is
            month: number where >= 1 and <= 12
            "/"
            day: number where integer and >= 1 and <= 31
            "/197"
            year: number where integer and >= 5 and <= 9
            where (if (month = 4 or = 5 or = 9 or = 11) then day <= 30)
                  and (if month = 2 then day <= 29)
                  and (if (month = 2 and year ~= 6) then day <= 28)
      ordering
            year, month, day

COUNTRY_NAMES
    derivation: subset of STRINGS
```

Figure 3-7. Syntax of the Data Definition Language (DDL)

Notes:

- The left side of a production is separated from the right by a "<--".

- The first level of indentation in the syntax description is used to help separate the left and right sides of a production; all other indentation is in the SDM DDL.

- Syntactic categories are capitalized, while all literals are in lower case.

- Symbols:

```
{ }   - means optional
[ ]   - means one of the enclosed choices must appear; choices
            are separated by a ";"
            (when used with "{}" one of the choices may optionally
            appear)
< >   - means one or more of the enclosed can appear, separated
            by spaces with optional commas and an optional "and" at
            the end
<< >> - means one or more of the enclosed can appear, vertically
            appended
* *   - encloses a "meta"-description of a syntactic category
            (to informally explain it)
```

```
SCHEMA <--
    <<CLASS>>

CLASS <--
    CLASS_NAME
        {description:  CLASS_DESCRIPTION}
        {[BASE_CLASS_FEATURES; CLASS_DERIVATION]}
        {MEMBER_ATTRIBUTES}
        {CLASS_ATTRIBUTES}
        {CLASS-DETERMINED_ATTRIBUTES}
```

Figure 3-7. Syntax of the SDM DDL (continued)

```
CLASS_NAME <--
    *string of capitals possibly including '_'*

CLASS_DESCRIPTION <--
    "*string*"

BASE_CLASS_FEATURES <--
    {kind: CLASS_KIND}
    {[duplicates allowed; duplicates not allowed]}
    {IDENTIFIERS}

CLASS_KIND <--
    [{concrete} object; {[duration; point]} event; name]

IDENTIFIERS <--
    <ATTRIBUTE_NAME>

MEMBER_ATTRIBUTES <--
    member attributes:
        <<MEMBER_ATTRIBUTE>>

CLASS_ATTRIBUTES <--
    class attributes:
        <<CLASS_ATTRIBUTE>>

CLASS-DETERMINED_ATTRIBUTES <--
    class-determined attributes:
        <<CLASS-DETERMINED_ATTRIBUTE>>

CLASS_DERIVATION <--
    [RESTRICT; SUBSET; EXTRACT_COMMON_MEMBERS;
    MERGE_MEMBERS; EXTRACT_MISSING_MEMBERS;
    ABSTRACT; AGGREGATE]
```

Figure 3-7. Syntax of the SDM DDL (continued)

```
RESTRICT <--
    restrict CLASS_NAME {where RESTRICT_PREDICATE}

SUBSET <--
    subset of CLASS_NAME

EXTRACT_COMMON_MEMBERS <--
    extract common members in <CLASS_NAME>

MERGE_MEMBERS <--
    merge members in <CLASS_NAME>

EXTRACT_MISSING_MEMBERS <--
    extract missing members in CLASS_NAME but not in CLASS_NAME

ABSTRACT <--
    abstract CLASS_NAME on common value of <ATTRIBUTE_NAME>
    {defined instances classes are <CLASS_NAME>}

AGGREGATE <--
    [PRIMITIVE_AGGREGATE; DERIVED_AGGREGATE;
    MIXED_AGGREGATE]

PRIMITIVE_AGGREGATE <--
    aggregate of CLASS_NAME

DERIVED_AGGREGATE <--
    derived aggregate of CLASS_NAME
    {defined constituents classes are <CLASS_NAME>}

MIXED_AGGREGATE <--
    mixed aggregate of CLASS_NAME
    {defined constituents classes are <CLASS_NAME>}
```

Figure 3-7. Syntax of the SDM DDL (continued)

```
RESTRICT_PREDICATE <--
    [SIMPLE_PREDICATE; (RESTRICT_PREDICATE);
    not RESTRICT_PREDICATE;
    RESTRICT_PREDICATE and RESTRICT_PREDICATE;
    RESTRICT_PREDICATE or RESTRICT_PREDICATE;
    PATTERN]

SIMPLE_PREDICATE <--
    [MAPPING SCALAR_COMPARATOR [CONSTANT; MAPPING];
    MAPPING SET_COMPARATOR [CONSTANT; CLASS_NAME; MAPPING];
    is a value of ATTRIBUTE_NAME of CLASS_NAME]

MAPPING <--
    [ATTRIBUTE_NAME; MAPPING.ATTRIBUTE_NAME]

SCALAR_COMPARATOR <--
    [EQUAL_COMPARATOR; >; >=; <; <=]

EQUAL_COMPARATOR <--
    [=, ~=]

SET_COMPARATOR <--
    [is {properly} in; is not {properly} in;
    {properly} contains; does not {properly} contain]

CONSTANT <--
    ☼a string or number constant☼

PATTERN <--
    ☼a name class definition pattern☼
    (see Figure 3-6 and [McLeod 1977a])
```

Figure 3-7. Syntax of the SDM DDL (continued)

```
MEMBER_ATTRIBUTE <--
    ATTRIBUTE_NAME
        ATTRIBUTE_FEATURES
        [PRIMITIVE_ATTRIBUTE_FEATURES;
        DERIVED_MEMBER_ATTRIBUTE_FEATURES]

CLASS-DETERMINED_ATTRIBUTE <--
    ATTRIBUTE_NAME
        ATTRIBUTE_FEATURES
        [PRIMITIVE_ATTRIBUTE_FEATURES;
        DERIVED_CLASS-DETERMINED_ATTRIBUTE_FEATURES]

CLASS_ATTRIBUTE <--
    ATTRIBUTE_NAME
        ATTRIBUTE_FEATURES
        [PRIMITIVE_ATTRIBUTE_FEATURES;
        DERIVED_CLASS_ATTRIBUTE_FEATURES]

ATTRIBUTE_NAME <--
    *string of lower case letters beginning with a capital,
    and possibly including '_'*

ATTRIBUTE_FEATURES <--
    {ATTRIBUTE_DESCRIPTION}
    {ATTRIBUTE_KIND}
    {[[multi; class]-valued; single-valued]}

ATTRIBUTE_DESCRIPTION <--
    "*string*"

ATTRIBUTE_KIND <--
    [property; component; participant]
```

Figure 3-7. Syntax of the SDM DDL (continued)

```
PRIMITIVE_ATTRIBUTE_FEATURES <--
    VALUE_CLASS
    {[mandatory; optional]}
    {[fixed; changeable]}

VALUE_CLASS <--
    value class: CLASS_NAME

DERIVED_MEMBER_ATTRIBUTE_FEATURES <--
    derivation: MEMBER_ATTRIBUTE_DERIVATION

DERIVED_CLASS-DETERMINED_ATTRIBUTE_FEATURES <--
    derivation: CLASS-DETERMINED_ATTRIBUTE_DERIVATION

DERIVED_CLASS_ATTRIBUTE_FEATURES <--
    derivation: CLASS_ATTRIBUTE_DERIVATION

MEMBER_ATTRIBUTE_DERIVATION <--
    [INTER-ATTRIBUTE_DERIVATION;
    MEMBER-SPECIFIC_DERIVATION]

CLASS-DETERMINED_ATTRIBUTE_DERIVATION <--
    INTER-ATTRIBUTE_DERIVATION

CLASS_ATTRIBUTE_DERIVATION <--
    [INTER-ATTRIBUTE_DERIVATION;
    CLASS-SPECIFIC_DERIVATION]
```

Figure 3-7. Syntax of the SDM DDL (continued)

```
INTER-ATTRIBUTE_DERIVATION <--
    [same as MAPPING;
    = MAPPING_EXPRESSION;
    extract common members in <MAPPING>;
    merge members in <MAPPING>;
    extract missing members in MAPPING but not in MAPPING;
    [maximum, minimum, average, sum] of MAPPING;
    number of {unique} members in MAPPING;
    restrict ATTRIBUTE_NAME {where RESTRICT_PREDICATE}]

MEMBER-SPECIFIC_DERIVATION <--
    [order by [increasing, ascending, descreasing,
    descending] <MAPPING> {within <MAPPING>};
    invert ATTRIBUTE_NAME of CLASS_NAME;
    ATTRIBUTE_NAME of match to CLASS_NAME on ATTRIBUTE_NAME;
    merge members in <ATTRIBUTE_NAME> of match to CLASS_NAME
    on ATTRIBUTE_NAME;
    if exists in CLASS_NAME;
    merge members in repeat over ATTRIBUTE_NAME;
    instances;
    constituents]

CLASS-SPECIFIC_DERIVATION <--
    [number of {unique} members in this class;
    [maximum, minimum, average, sum] of ATTRIBUTE_NAME
    over members of this class]

MAPPING_EXPRESSION <--
    [MAPPING; (MAPPING); MAPPING NUMBER_OPERATOR MAPPING]

NUMBER_OPERATOR <--
    [+; -; *; /; !]
```

Figure 4-1. The Result of Designing an SDM Schema for the AMAE

```
MECHANICS
    description: all employees who are concerned directly or
        indirectly with aircraft maintenance
    kind: concrete object
    member attributes:
        Id
            value class: ID_NUMBERS
        Name
            value class: PERSON_NAMES
        Aircraft_types_qualified_on
            value class: AIRCRAFT_TYPES
            multi-valued
        Home_airport
            value class: AIRPORTS
    identifiers:
        Id

AIRPORTS
    kind: concrete object
    member attributes:
        Airport_id
            value class: AIRPORT_NAMES
        Mechanics_based_here
            derivation: invert Home_airport of MECHANICS
            multi-valued
        Number_of_mechanics_based_here
            derivation: number of members in Mechanics_base_here
    identifiers:
        Airport_id
```

Figure 4-1. An SDM Schema for the AMAE (continued)

```
AIRCRAFT
    kind: concrete object
    member attributes:
        Tail_number
            value class: AIRCRAFT_NUMBERS
        Type
            value class: AIRCRAFT_TYPE_NAMES
        Jobs
            description: all jobs performed on the aircraft
            derivation: invert Aircraft of JOBS
            multi-valued
        Scheduled_jobs
            derivation: invert Aircraft of SCHEDULED_JOBS
            multi-valued
        Next_scheduled_job
            derivation: restrict Scheduled_jobs where
                Order_for_aircraft = 1
    class attributes:
        Number
            derivation: number of members in this class
    identifiers:
        Tail_number

AIRCRAFT_TYPES
    description: kinds of aircraft, e.g., B707, D10, etc.
    derivation: abstract AIRCRAFT on common value of Type
    member attributes:
        Engine_type
            value class: ENGINE_TYPE_NAMES
        Number_of_engines
            value class: NUMBERS
```

Figure 4-1. An SDM Schema for the AMAE (continued)

```
AIRCRAFT_DUE_FOR_SERVICE
    derivation: subset of AIRCRAFT

MECHANIC_QUALIFICATION_GROUPS
    description: groups of employees by qualification
    derivation: abstract MECHANICS on common value of
        Aircraft_types_qualified_on
    member attributes:
        Number_of_employees_in_this_group
            derivation: number of members in Instances

JOBS
    description: maintenance actions on aircraft
    kind: event
    member attributes:
        Aircraft
            kind: participant
            value class: AIRCRAFT
        Date_scheduled_for
            value class: DATES
        Task
            value class: TASK_NAMES
        Job_number
            value class: JOB_NUMBERS
        Status
            value class: JOB_STATUS_NAMES
    identifiers:
        Job_number
```

Figure 4-1. An SDM Schema for the AMAE (continued)

```
SCHEDULED_JOBS
    derivation: restrict JOBS where Status = 'scheduled'
    member attributes:
        Order
            description: the order of the job over time
            derivation: order by ascending Date_scheduled_for
        Order_for_aircraft
            description: the order of the job over time for
                as given aircraft
            derivation: order by ascending Date_scheduled_for
                within Aircraft

JOBS_PERFORMED
    derivation: restrict JOBS where Status = 'complete'
    member attributes:
        Mechanics_performing
            kind: participant
            value class: MECHANICS
            multi-valued
        Date_performed
            value class: DATES
        Parts_used
            description: the types of parts used in the job
            value class: PART_TYPES
            multi-valued

BREAKDOWN_JOBS
    description: jobs in response to aircraft breakdowns
    derivation: restrict JOBS where Status = 'emergency'
    member attributes:
        Breakdown
            description: the breakdown causing the repair job
            value class: BREAKDOWNS
```

Figure 4-1. An SDM Schema for the AMAE (continued)

```
BREAKDOWNS
    description: aircraft breakdowns
    kind: event
    member attributes:
        Aircraft
            kind: participant
            value class: AIRCRAFT
        Date
            value class: DATES
        Problem
            value class: PROBLEMS
        Job
            description: the job fixing the breakdown
            derivation: invert Breakdown of BREAKDOWN_JOBS

PART_TYPES
    description: types of parts for aircraft repair
    kind: object
    member attributes:
        Part_number
            value class: PART_NUMBERS
        Part_description
            value class: TEXT
        Subparts
            description: the parts used in this part
            value class: PART_TYPES
            multi-valued
        All_subparts
            description: the parts used in this part or a
                subpart of it
            derivation: merge members in repeat over Subparts
            multi-valued
    identifiers:
        Part_number
```

Figure 4-1. An SDM Schema for the AMAE (continued)

```
PART_STOCKAGES_BY_AIRPORT
    description: the parts on hand at an airport
    kind: event
    member attributes:
        Part_type
            value class: PART_TYPES
        Location
            description: the airport stocking the part
            value class: AIRPORTS
        Quantity_on_hand
            value class: NUMBERS

PART_OUTTAGES
    description: occurrences of airports running out of parts
        of a given type
    derivation: restrict PART_STOCKAGES_BY_AIRPORT where
        Quantity_on_hane = 0

PART_TYPES_OUT_OF_STOCK_SOMEWHERE
    description: parts that are out of stock at some airport
    derivation: restrict PART_TYPES where is a value of
        Part_type of PART_OUTTAGES

ID_NUMBERS
    derivation: restrict NUMBERS

PERSON_NAMES
    derivation: restrict STRINGS

AIRPORT_NAMES
    derivation: restrict STRINGS

AIRCRAFT_NUMBERS
    derivation: restrict STRINGS
```

Figure 4-1. An SDM Schema for the AMAE (continued)

```
AIRCRAFT_TYPE_NAMES
    derivation: restrict STRINGS

ENGINE_TYPE_NAMES
    derivation: restrict STRINGS

AIRCRAFT_STATUS_LEVELS
    derivation: restrict STRINGS

DATES
    derivation: restrict STRINGS

TASK_NAMES
    derivation: restrict STRINGS

JOB_NUMBERS
    derivation: restrict NUMBERS

JOB_STATUS_NAMES
    derivation: restrict STRINGS

PART_NUMBERS
    derivation: restrict NUMBERS

TEXT
    derivation: restrict STRINGS

PROBLEMS
    derivation: restrict STRINGS
```

Figure 4-2. An Example of Multiple Derivation Specifications for a Class

```
                        base concrete
                        object class
                            SHIPS
                              |
       ----------------------------------------------
       |                      |                     |
    restrict              restrict              restrict
 (where Cargo_types     (where Type         (where Cargo_types
   contains 'oil'       = 'military')       contains 'oil' and
       |                      |              Type = 'military)
       v                      v                     |
   OIL_TANKERS          MILITARY_SHIPS              |
       |                      |                     |
    restrict              restrict                  |
  (where Type         (where Cargo_types            |
  = 'military'          contains 'oil')             |
       |                      |                     |
       ----------------------------------------------
                              |
                              v
                    MILITARY_OIL_TANKERS
```

Figure 6-1. Overview of a Possible SDM DBMS

```
                  User
                   |
      -------------------------
      |            |          |
      |            v          v
      |      ----------------
      |      | Applications |
      |      |    System    |
      |      ----------------
      |            |
      v            |
 ---------------   |
 | Interaction |   |
 | Formulation |   |
 |   Advisor   |   |
 ---------------   |
      |            |
      --------------
      | user interface to SDM
      | (interaction formalism)
      v
 -------------------------
 |  Semantic Data Model  |
 |-----------------------|
 |  SDM data base - 1    |
 |  SDM data base - 2    |
 |       (etc.)          |
 -------------------------
           |
           v
 ---------------------------
 |    Conventional DBMS    |
 |    (e.g., relational)   |
 |-------------------------|
 | Underlying data base - 1|
 | Underlying data base - 2|
 |         (etc.)          |
 ---------------------------
```

Appendix 1-a. Transcript of an Example IFAP Interaction

```
Interaction Formulation Advisor Prototype

Welcome to the Interaction Formulation Advisor. Please
terminate all responses (everything you type) with a
carriage return (CR). If you need assistance at any time,
type a question mark (?). Type an exclamation point (!) to
stop and start over.

The following data base(s) are available:
    1. TMDB SCHEMA: concerning monitoring ships with hazardous
       cargoes
    2. none of the above
Which of these would you like to use?
Please enter the appropriate number.
> 1
The schema for the SDM data base in file TMDB SCHEMA is
being loaded.

Do you want a record of this interaction? (Please type yes
or no, or y or n. End your response with a carriage return
(CR).)
Please answer yes or no.
> no


S1. Are you experienced in the use of the Interaction
Formulation Advisor?
Please answer yes or no.
> ?
Let's assume that you are inexperienced.
The purpose of this interaction is to extract some
information from the data base. The process you will
follow has two steps.
(1)
The point of the first step in the interaction formulation
process is to identify the collection of things (objects,
events, names, etc.) in which you are interested. Such a
collection is called a class and the class in which you are
interested will be called the target class. It may be the
```

case that the target class is defined (known) in the data base. If so, you can select it by giving its name. If not, you will be called upon to assist in describing it, i.e., by providing a description of it in terms of defined classes.

(2)

Having identified the target class, the second step in the interaction formulation process asks you to specify the aspect(s) of the class in which you are interested. You will have the option of examining one or more attributes of each member of the target class, or of the target class as a whole. The attribute(s) you desire will be called the target attributes. As is the case for classes, the target attributes may or may not be known in the data base; if they are not known, you will need to help in defining them.

The first task is to identify the target class.

S2. Can you name a class defined in the data base, so that the class is the target class?
Please answer yes or no.
> no

S3. Do you know the name of a relevant class which is defined in the data base?
Please answer yes or no.
> ?
A relevant class, also called a class of interest, is one that is closer to the target class than the working class. For example, if you are interested in the class of all female computer programmers, the class of all computer programmers is relevant. The class of all people is relevant also, but not as close. Normally, if you are seeking a very specific class, you will find it through several steps of classes that are closer and closer to the target class.

S4. Do you know the name of a relevant class which is defined in the data base?
Please answer yes or no.
> no

S5. Although you don't know which one, perhaps one of the classes defined in the data base is a relevant class, or even the target class. The following basic classes are defined:

1. SHIPS (all ships with potentially hazardous cargoes that may enter U.S. coastal waters)
2. INSPECTIONS (inspections of oil tankers)
3. COUNTRIES
4. OFFICERS (all certified officers of ships)
5. ENGINES
6. INCIDENTS (ship accidents)
7. ASSIGNMENTS (assignments of captains to ships)
8. none of the above

Which one of these is most relevant?
Please enter the appropriate number.
> 1

S6. Is SHIPS the target class?
Please answer yes or no.
> no

S7. Do you think that a class which contains some (but probably not all) of the members of SHIPS is the target class or a relevant class?
Please answer yes or no.
> ?
A class that contains some of the members of another is called a subclass. A class that contains all of the members of another, and probably more, is called a superclass.

S8. Do you think that a class which contains some (but probably not all) of the members of SHIPS is the target class or a relevant class?
Please answer yes or no.
> yes

S9. The subclasses of SHIPS that are defined are:

1. OIL_TANKERS
2. MERCHANT_SHIPS
3. BANNED_SHIPS (ships banned from U.S. coastal waters)
4. SAFE_SHIPS (ships which are considered good risks)

```
    5. none of the above
Which one of these is relevant?
Please enter the appropriate number.
> 1

S10. Is OIL_TANKERS the target class?
Please answer yes or no.
> no

S11. Do you think that a class which contains some (but probably
not all) of the members of OIL_TANKERS is the target class
or a relevant class?
Please answer yes or no.
> yes

S12. The subclasses of OIL_TANKERS that are defined are:
    1. LIBERIAN_OIL_TANKERS
    2. OIL_TANKERS_REQUIRING_INSPECTION
    3. BANNED_OIL_TANKERS
    4. none of the above
Which one of these is relevant?
Please enter the appropriate number.
> 1

S13. Is LIBERIAN_OIL_TANKERS the target class?
Please answer yes or no.
> yes

Now that the target class has been determined, the next
step is to describe the target attributes.
In order to actually examine information in the data base,
it is necessary for you to specify which aspects of the
target class you find relevant. You do this by selecting
attributes of each member of the target class, or of the
target class as a whole. You will first be asked to select
attributes that have a value for each member of the target
class. You will then be asked to choose attributes of the
class as a whole and those which have the same value for
each member of the target class.
First, you can select aspects (attributes) of each member
of LIBERIAN_OIL_TANKERS in which you are interested.
```

S14. Are you interested in examining attribute(s) of each member of LIBERIAN_OIL_TANKERS?
Please answer yes or no.
> yes

S15. The following member attributes of LIBERIAN_OIL_TANKERS are defined:

1. Oil_spills_involved_in
2. Hull_type (specification of single or double hull)
3. Is_tanker_banned?
4. Inspections
5. Number_of_times_inspected
6. Last_inspection
7. Last_two_inspections
8. Date_last_examined
9. Name
10. Hull_number
11. Type (the kind of ship, e.g., merchant or fishing)
12. Country_of_registry
13. Name_of_home_port
14. Cargo_types (the type(s) of cargo the ship can carry)
15. Captain (the current captain of the ship)
16. Engines
17. Oil-burning_engines
18. none of the above

Which one of these is relevant? If several are of interest, please choose one now, and this request will repeat.
Please enter the appropriate number.
> 9

S16. Are you interested in any other member attributes of LIBERIAN_OIL_TANKERS?
Please answer yes or no.
> yes

S17. The following member attributes of LIBERIAN_OIL_TANKERS are defined:

1. Oil_spills_involved_in
2. Hull_type (specification of single or double hull)
3. Is_tanker_banned?
4. Inspections
5. Number_of_times_inspected

```
     6. Last_inspection
     7. Last_two_inspections
     8. Date_last_examined
     9. Name
     10. Hull_number
     11. Type (the kind of ship, e.g., merchant or fishing)
     12. Country_of_registry
     13. Name_of_home_port
     14. Cargo_types (the type(s) of cargo the ship can carry)
     15. Captain (the current captain of the ship)
     16. Engines
     17. Oil-burning_engines
     18. none of the above
Which one of these is relevant? If several are of interest,
please choose one now, and this request will repeat.
Please enter the appropriate number.
> 15

S18. Attribute Captain of class LIBERIAN_OIL_TANKERS cannot be
directly printed, because it is not in the form of a number
or a string of characters. Values of Captain are members
of the class OFFICERS. Thus, you may be able to print the
value of an attribute of the value(s) of Captain of class
LIBERIAN_OIL_TANKERS. Would you like to try to do this?
Please answer yes or no.
> ?
Some atributes have values that can be printed or typed,
while other attribute values cannot be directly examined or
entered on a keyboard. Specifically, only numbers and
character strings can be printed and typed in. This means
that in order to print the value of an attribute or compare
the value of an attribute with some item you type in, that
value must be a string or a number. Thus, for example,
entities like students and textbooks probably cannot be
printed, while textbook names and student addresses
probably are printable. Of course, what is printable
depends on the way in which a given data base is defined.
If you want to print the value ofan attribute that is not
printable, will be asked if you wish to print an attribute
of the values of that attribute. Similarly, if you want to
compare the value of an attribute with a value you intend
to type in and that attribute is not printable, you will be
```

asked if you want to compare the item to be typed in with the value of an attribute of the attribute. For example, if you select the attribute Textbook of a class COURSES_OFFERED, you will be reminded that you cannot examine that attribute directly, and asked to choose an attribute of TEXTBOOKS, the class containing values of the attribute Textbook.

S19. Attribute Captain of class LIBERIAN_OIL_TANKERS cannot be directly printed, because it is not in the form of a number or a string of characters. Values of Captain are members of the class OFFICERS. Thus, you may be able to print the value of an attribute of the value(s) of Captain of class LIBERIAN_OIL_TANKERS. Would you like to try to do this?
Please answer yes or no.
> yes

S20. Values of Captain are members of the class OFFICERS. The attributes of members of OFFICERS are as follows:

1. Name
2. Country_of_license
3. Date_commissioned
4. Commander (the officer in direct command of this officer)
5. Superiors
6. Subordinates
7. none of the above

Which one of these would you like to examine? If more than one is relevant, please pick one now, and this request will repeat.
Please enter the appropriate number.
> 1

S21. Having focused on attribute Captain of LIBERIAN_OIL_TANKERS, you can choose any number of attributes of members of OFFICERS. You have already selected at least one; are there any others?
Please answer yes or no.
> no

S22. Are you interested in any other member attributes of LIBERIAN_OIL_TANKERS?
Please answer yes or no.

```
> no
The target member attributes are:
    1. Name
    2. Name of Captain

S23. Are you happy with this?
Please answer yes or no.
> yes
Now, you can select aspects (attributes) of the class
LIBERIAN_OIL_TANKERS as a whole, and those which describe
each member of LIBERIAN_OIL_TANKERS and which have the same
value for each member.

S24. Are you interested in examining attribute(s) of the class
LIBERIAN_OIL_TANKERS as a whole or which have the same
value for each member of LIBERIAN_OIL_TANKERS?
Please answer yes or no.
> ?
An attribute associated with a class as a whole is one that
describes aspects of the target class taken as a unit, or
which has the same value for each member of the target
class. The former is specifically called a class
attribute; for example, a class attribute may contain as
its value the number of members in that class. The latter
is called a class-determined member attribute; for
example, the class waiters (a subclass of people, say)
might have the class-determined attribute that specifies
the maximum salary of each and every waiter.

S25. Are you interested in examining attribute(s) of the class
LIBERIAN_OIL_TANKERS as a whole or which have the same
value for each member of LIBERIAN_OIL_TANKERS?
Please answer yes or no.
> no

The transaction you have formulated is:

print member attributes of class LIBERIAN_OIL_TANKERS
    Name,
    Captain.Name
```

```
S26. Do you wish to formulate another interaction?
Please answer yes or no.
> no

S27. Do you want to exit from the Interaction Formulation
Advisor?
Please answer yes or no.
> yes
Thank you for your time.
```

Appendix 1-b. Another Transcript of an Example Interaction

```
S1. Are you experienced in the use of the Interaction
Formulation Advisor?
Please answer yes or no.
> y

The first task is to identify the target class.

S2. Can you name a class defined in the data base, so that the
class is the target class?
Please answer yes or no.
> n

S3. Do you know the name of a relevant class which is defined
in the data base?
Please answer yes or no.
> y

S4. What is its name?
Please type it in.
> Liberian oil tanker

S5. Is LIBERIAN_OIL_TANKERS the target class?
Please answer yes or no.
> n

S6. Do you think that a class which contains some (but probably
not all) of the members of LIBERIAN_OIL_TANKERS is the
target class or a relevant class?
Please answer yes or no.
> n

S7. Do you think that a class containing all of the members of
LIBERIAN_OIL_TANKERS (and probably more) is the target
class or a relevant class?
Please answer yes or no.
> y

S8. The superclasses of LIBERIAN_OIL_TANKERS that are defined
are:
    1. OIL_TANKERS
    2. none of the above
Which one of these is relevant?
Please enter the appropriate number.
> 1

S9. Is OIL_TANKERS the target class?
Please answer yes or no.
> n

S10. Do you think that a class which contains some (but probably
not all) of the members of OIL_TANKERS is the target class
or a relevant class?
Please answer yes or no.
> y

S11. The subclasses of OIL_TANKERS that are defined are:
    1. LIBERIAN_OIL_TANKERS
    2. OIL_TANKERS_REQUIRING_INSPECTION
    3. BANNED_OIL_TANKERS
    4. none of the above
Which one of these is relevant?
Please enter the appropriate number.
> 4

S12. Are you sure that you are interested in a class that
contains some of the members of OIL_TANKERS?
Please answer yes or no.
> y

S13. The class in which you are interested, that contains some
of the members of OIL_TANKERS is apparently not defined in
the data base. Would you like to try to define this
subclass?
Please answer yes or no.
> ?
If the target class is not defined in the data base, you
can define it yourself. You do this by defining this new
class so that it relates to another class defined in the
data base in one of the following ways:

    - the new class contains some of the members of the
```

other class
- the new class contains all of the members of the other class, and contains additional members
- the new class contains all of the members of some class that are not in the other class
- the new class contains abstractions (types or groups) of the members of the other class

S14. The class in which you are interested, that contains some of the members of OIL_TANKERS is apparently not defined in the data base. Would you like to try to define this subclass?
Please answer yes or no.
> y

S15. One way to define a subclass is to specify another class in the data base, so that the target class or a relevant class contains precisely the members that this other class shares with OIL_TANKERS. Do you think this is possible?
Please answer yes or no.
> y

S16. The following classes that may have members in common with OIL_TANKERS are defined:

1. MERCHANT_SHIPS
2. BANNED_SHIPS (ships banned from U.S. coastal waters)
3. SAFE_SHIPS (ships which are considered good risks)
4. SHIPS_TO_BE_MONITORED (ships which are considered bad risks)
5. none of the above

Which one of them will yield a relevant class or the target class when the members it shares with OIL_TANKERS are determined?
Please enter the appropriate number.
> 1

S17. Is MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS] the target class?
Please answer yes or no.
> n

S18. Do you think that a class which contains some (but probably not all) of the members of

```
MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS] is the
target class or a relevant class?
Please answer yes or no.
> y

S19. Are you sure that you are interested in a class that
contains some of the members of
MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS]?
Please answer yes or no.
> y

S20. The class in which you are interested, that contains some
of the members of
MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS] is
apparently not defined in the data base. Would you like to
try to define this subclass?
Please answer yes or no.
> y

S21. One way to define a subclass is to specify another class in
the data base, so that the target class or a relevant class
contains precisely the members that this other class shares
with MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS].
Do you think this is possible?
Please answer yes or no.
> n

S22. You can define a subclass by specifying another class in
the data base, so that the target class or a relevant class
contains precisely the members of
MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS] that are
not in this other class. Do you think this is possible?
Please answer yes or no.
> y

S23. The following classes may be helpful:
    1. LIBERIAN_OIL_TANKERS
    2. BANNED_SHIPS (ships banned from U.S. coastal waters)
    3. OIL_TANKERS_REQUIRING_INSPECTION
    4. BANNED_OIL_TANKERS
    5. SAFE_SHIPS (ships which are considered good risks)
    6. SHIPS_TO_BE_MONITORED (ships which are considered bad risks)
```

7. none of the above
Which one of them will yield a relevant class or the target class when its members are removed from MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS]?
Please enter the appropriate number.
> 7

S24. You can describe a new class which contains some, but not all, of the members of MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS] by specifying some constraint(s) on the value of the attributes of members of MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS]. Does this sound reasonable?
Please answer yes or no.
> ?
It may be possible to define a new subclass of the working class by specifying some constraint a member of the working class must satisfy in order for it to be a member of the new class. The constraint is specified by stating a limitation on the value of member attribute(s) of the working class. For example, suppose the data base contains the class STUDENTS. And suppose that STUDENTS has the member attribute Sex. Then, to define the class of all male students, one can state that MALE_STUDENTS is the restriction of STUDENTS where attribute Sex has the particular value male.

S25. You can describe a new class which contains some, but not all, of the members of MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS] by specifying some constraint(s) on the value of the attributes of members of MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS]. Does this sound reasonable?
Please answer yes or no.
> y

S26. The primary goal now is to choose an attribute of members of MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS]; then you can state a constraint on values of that attribute. The following member attributes of

```
MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS] are
defined:
    1. Hull_type (specification of single or double hull)
    2. Is_tanker_banned?
    3. Inspections
    4. Number_of_times_inspected
    5. Last_inspection
    6. Last_two_inspections
    7. Date_last_examined
    8. Name
    9. Hull_number
    10. Type (the kind of ship, e.g., merchant or fishing)
    11. Country_of_registry
    12. Name_of_home_port
    13. Cargo_types (the type(s) of cargo the ship can carry)
    14. Captain (the current captain of the ship)
    15. Engines
    16. Oil-burning_engines
    17. none of the above
You need to pick one of these attributes, so that you can
also specify some constraint its value must have in order
for a member of
MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS] to be a
member of the class being defined. If more than one
attribute is involved, you should pick the most important
one; then later, if you want, you can impose additional
constraints when this request repeats.
Please enter the appropriate number.
> 11
First, let's see if you want to compare Country_of_registry
with some value you can type in.

S27. Would you like to try to state this kind of constraint on
an attribute of the value of Country_of_registry?
Please answer yes or no.
> y

S28. Values of Country_of_registry of
MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS] are
members of the class COUNTRIES. The attributes of members
of COUNTRIES are as follows:
    1. Name
```

2. Ships_registered_here
3. none of the above

Which of these is relevant?
Please enter the appropriate number.
> 1

S29. Do you wish to require the value of Name to be equal to, not equal to, less than, less than or equal to, greater than, or greater than or equal to some value you want to type in?
Please answer yes or no.
> y

S30. Several kinds of comparisons are possible. (Less than and such for character strings means according to dictionary ordering.):

1. equal to
2. not equal to
3. greater than
4. greater than or equal to
5. less than
6. less than or equal to
7. *none of the above*

Which one is appropriate?
Please enter the appropriate number.
> 1

S31. Can you type in the value that Name must be equal to?
Please answer yes or no.
> y

S32. Enter the value that Name must be equal to.
Please type it in.
> Panama

S33. *You have specified a constraint on the members of* MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS], by defining the class [MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS]]_WHERE_ [COUNTRY_OF_REGISTRY.NAME_=_"PANAMA"].
In a short while, you can further constrain this class if you want, i.e., by defining a class which has some of its

members, or a class which contains all of its members plus
others. The only issue now is to make sure that
[MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS]]_WHERE_
COUNTRY_OF_REGISTRY.NAME_=_"PANAMA"] is relevant. Is it?
Please answer yes or no.
> y

S34. Is
[MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS]]_WHERE_
[COUNTRY_OF_REGISTRY.NAME_=_"PANAMA"] the target class?
Please answer yes or no.
> y

Now that the target class has been determined, the next
step is to describe the target attributes.
First, you can select aspects (attributes) of each member
of [MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS]]_WHERE_
[COUNTRY_OF_REGISTRY.NAME_=_"PANAMA"] in which you are interested.

S35. Are you interested in examining attribute(s) of each member
of [MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS]]_WHERE_
[COUNTRY_OF_REGISTRY.NAME_=_"PANAMA"]?
Please answer yes or no.
> y

S36. The following member attributes of
[MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS]]_WHERE_
[COUNTRY_OF_REGISTRY.NAME_=_"PANAMA"] are defined:

1. Hull_type (specification of single or double hull)
2. Is_tanker_banned?
3. Inspections
4. Number_of_times_inspected
5. Last_inspection
6. Last_two_inspections
7. Date_last_examined
8. Name
9. Hull_number
10. Type (the kind of ship, e.g., merchant or fishing)
11. Country_of_registry
12. Name_of_home_port
13. Cargo_types (the type(s) of cargo the ship can carry)
14. Captain (the current captain of the ship)

```
     15. Engines
     16. Oil-burning_engines
     17. none of the above
Which one of these is relevant? If several are of interest,
please choose one now, and this request will repeat.
Please enter the appropriate number.
> 9

S37. Are you interested in any other member attributes of
[MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS]]_WHERE_
[COUNTRY_OF_REGISTRY.NAME_=_"PANAMA"]?
Please answer yes or no.
> y

S38. The following member attributes of
[MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS]]_WHERE_
[COUNTRY_OF_REGISTRY.NAME_=_"PANAMA"] are defined:
     1. Hull_type (specification of single or double hull)
     2. Is_tanker_banned?
     3. Inspections
     4. Number_of_times_inspected
     5. Last_inspection
     6. Last_two_inspections
     7. Date_last_examined
     8. Name
     9. Hull_number
     10. Type (the kind of ship, e.g., merchant or fishing)
     11. Country_of_registry
     12. Name_of_home_port
     13. Cargo_types (the type(s) of cargo the ship can carry)
     14. Captain (the current captain of the ship)
     15. Engines
     16. Oil-burning_engines
     17. none of the above
Which one of these is relevant? If several are of interest,
please choose one now, and this request will repeat.
Please enter the appropriate number.
> 7

S39. Are you interested in any other member attributes of
[MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS]]_WHERE_
[COUNTRY_OF_REGISTRY.NAME_=_"PANAMA"]?
```

```
Please answer yes or no.
> n
The target member attributes are:
    1. Hull_number
    2. Date_last_examined

S40. Are you happy with this?
Please answer yes or no.
> y
Now, you can select aspects (attributes) of the class
[MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS]]_WHERE_
[COUNTRY_OF_REGISTRY.NAME_=_"PANAMA"] as a whole, and those
which describe each member of
[MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS]]_WHERE_
[COUNTRY_OF_REGISTRY.NAME_=_"PANAMA"] and which have the same
value fcr each member.

S41. Are you interested in examining attribute(s) of the class
[MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS]]_WHERE_
[COUNTRY_OF_REGISTRY.NAME_=_"PANAMA"] as a whole or which have
the same value for each member of
[MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS]]_WHERE_
[COUNTRY_OF_REGISTRY.NAME_=_"PANAMA"]?
Please answer yes or no.
> n

The transaction you have formulated is:

MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS] <-
    extract common members in OIL_TANKERS and MERCHANT_SHIPS

[MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS]]_WHERE_
    [COUNTRY_OF_REGISTRY.NAME_=_"PANAMA"]
    <- restrict
    MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS] where
    Country_of_registry.Name = "Panama"

print member attributes of class
    [MEMBERS_IN_BOTH_[OIL_TANKERS]_AND_[MERCHANT_SHIPS]]_WHERE_
    [COUNTRY_OF_REGISTRY.NAME_=_"PANAMA"]
    Hull_number,
    Date_last_examined
```

Appendix 1-c. Another Transcript of an Example Interaction

```
S1. Are you experienced in the use of the Interaction
Formulation Advisor?
Please answer yes or no.
> y

The first task is to identify the target class.

S2. Can you name a class defined in the data base, so that the
class is the target class?
Please answer yes or no.
> y

S3. What is its name?
Please type it in.
> merchant ships

S4. Is MERCHANT_SHIPS the target class?
Please answer yes or no.
> n

S5. Do you think that a class which contains some (but probably
not all) of the members of MERCHANT_SHIPS is the target
class or a relevant class?
Please answer yes or no.
> n

S6. Do you think that a class containing all of the members of
MERCHANT_SHIPS (and probably more) is the target class or a
relevant class?
Please answer yes or no.
> n

S7. Perhaps you can identify a class defined in the data base
so that when the members of MERCHANT_SHIPS are removed from
it, you obtain the target class or a relevant class. Is
this the case?
Please answer yes or no.
> n
```

```
S8. Perhaps you are interested in a class that contains types
(abstractions) of MERCHANT_SHIPS. Might such an
abstraction class be the target class or a relevant class?
Please answer yes or no.
> n

S9. MERCHANT_SHIPS is a class containing members whose type is
a member of a class of types (abstractions): SHIP_TYPES.
Is SHIP_TYPES the target class or a relevant class?
Please answer yes or no.
> ?
An abstraction class is one whose members are types or
groups of the members of another. For example, if one has
a class whose members are people, there might be a class
defined whose members are types of people (e.g., employees,
students, etc.). The instances of the member of an
abstraction class are the things of that type (or in that
group); for example, the individual students are the
instances of the student type (i.e., of the member of the
class whose members are types of people).

S10. MERCHANT_SHIPS is a class containing members whose type is
a member of a class of types (abstractions): SHIP_TYPES.
Is SHIP_TYPES the target class or a relevant class?
Please answer yes or no.
> n

S11. You seem to be having trouble in identifying the target
class. Would you like the questions about MERCHANT_SHIPS
to repeat?
Please answer yes or no.
> y

S12. Is MERCHANT_SHIPS the target class?
Please answer yes or no.
> n

S13. Do you think that a class which contains some (but probably
not all) of the members of MERCHANT_SHIPS is the target
class or a relevant class?
Please answer yes or no.
> n
```

```
S14. Do you think that a class containing all of the members of
MERCHANT_SHIPS (and probably more) is the target class or a
relevant class?
Please answer yes or no.
> y

S15. The superclasses of MERCHANT_SHIPS that are defined are:
    1. SHIPS (all ships with potentially hazardous cargoes that
         may enter U.S. coastal waters)
    2. none of the above
Which one of these is relevant?
Please enter the appropriate number.
> 1

S16. Is SHIPS the target class?
Please answer yes or no.
> n

S17. Do you think that a class which contains some (but probably
not all) of the members of SHIPS is the target class or a
relevant class?
Please answer yes or no.
> n

S18. Perhaps you are interested in a class that contains types
(abstractions) of SHIPS. Might such an abstraction class
be the target class or a relevant class?
Please answer yes or no.
> n

S19. Perhaps you are interested in a class that contains members
that are collections (aggregates) of SHIPS. Might such an
aggregate class be the target class or a relevant class?
Please answer yes or no.
> ?
An aggregate class is one whose members are collections of
the members of another. For example, the class of all
project teams is an aggregate class; the members of this
class are collections of employees. The constituents of
the members of an aggregate class are the individual things
that belong to the collection. In the example, the
```

employees that constitute a given project team (member of the class of all project teams) are the constituents of that member (project team).

S20. Perhaps you are interested in a class that contains members that are collections (aggregates) of SHIPS. Might such an aggregate class be the target class or a relevant class?
Please answer yes or no.
> n

S21. Perhaps the target class (or a relevant class) is related to the working class indirectly. In particular, it may be the case that you are interested in a class that has an attribute whose values are members of SHIPS. Do you think this is the case?
Please answer yes or no.
> ?
Perhaps you are really interested in some other class in the data base which has an attribute whose values are members of the working class. For example, if the working class were TEXTBOOKS, perhaps you would really be interested in class COURSES_OFFERED, which has an attribute Text whose values are members of TEXTBOOKS.

S22. Perhaps the target class (or a relevant class) is related to the working class indirectly. In particular, it may be the case that you are interested in a class that has an attribute whose values are members of SHIPS. Do you think this is the case?
Please answer yes or no.
> y

S23. Below is a list all attributes defined in the data base which have values in class SHIPS.
1. INSPECTIONS: with attribute Tanker (of members)
2. COUNTRIES: with attribute Ships_registered_here (of members)
3. INCIDENTS: with attribute Involved_ship (of members)
4. ASSIGNMENTS: with attribute Ship (of members)
5. OIL_SPILLS: with attribute Involved_ship (of members)
6. MOST_RECENT_INSPECTIONS: with attribute Tanker (of members)
7. SHIP_TYPES: with attribute Instances (of members)

```
    8. CONVOYS: with attribute Oil_tanker_constituents (of
       members)
    9. CONVOYS: with attribute Constituents (of members)
   10. none of the above
Which one of the classes in the list is relevant?
Please enter the appropriate number.
> 1
Class INSPECTIONS is now the working class.

S24. Is INSPECTIONS the target class?
Please answer yes or no.
> n

S25. Do you think that a class which contains some (but probably
not all) of the members of INSPECTIONS is the target class
or a relevant class?
Please answer yes or no.
> y

S26. The subclasses of INSPECTIONS that are defined are:
    1. MOST_RECENT_INSPECTIONS
    2. none of the above
Which one of these is relevant?
Please enter the appropriate number.
> 2

S27. Are you sure that you are interested in a class that
contains some of the members of INSPECTIONS?
Please answer yes or no.
> y

S28. The class in which you are interested, that contains some
of the members of INSPECTIONS is apparently not defined in
the data base. Would you like to try to define this
subclass?
Please answer yes or no.
> y

S29. You can define a subclass by specifying another class in
the data base, so that the target class or a relevant class
contains precisely the members of INSPECTIONS that are not
in this other class. Do you think this is possible?
```

```
Please answer yes or no.
> n

S30. You can describe a new class which contains some, but not
all, of the members of INSPECTIONS by specifying some
constraint(s) on the value of the attributes of members of
INSPECTIONS. Does this sound reasonable?
Please answer yes or no.
> y

S31. The primary goal now is to choose an attribute of members
of INSPECTIONS; then you can state a constraint on values
of that attribute. The following member attributes of
INSPECTIONS are defined:
    1. Tanker (the tanker inspected)
    2. Date
    3. Order_for_tanker (the ordering of the inspections for a
       tanker with the most recent inspection having value 1)
    4. none of the above
You need to pick one of these attributes, so that you can
also specify some constraint its value must have in order
for a member of INSPECTIONS to be a member of the class
being defined. If more than one attribute is involved, you
should pick the most important one; then later, if you
want, you can impose additional constraints when this
request repeats.
Please enter the appropriate number.
> 1
First, let's see if you want to compare Tanker with some
value you can type in.

S32. Would you like to try to state this kind of constraint on
an attribute of the value of Tanker?
Please answer yes or no.
> n
Let's see if you want to constrain the value of Tanker to
be in (or not in) some other class.

S33. Perhaps you want to require the value of Tanker to be
present in another class. Do you?
Please answer yes or no.
> y
```

```
In that case, the next step is to specify the class in the
data base that the value of Tanker must be in.
In order to do this, you will have to identify this other
class;  let us call this the SECONDARY target class.  As
you will see, we will go through the same process of
finding a target class that we have been going through up
to now, except when we find this SECONDARY target class, we
will come back here and pick up where we left off.

S34.  Can you name a class defined in the data base, so that the
class is the target class?
Please answer yes or no.
> n

S35.  Do you know the name of a relevant class which is defined
in the data base?
Please answer yes or no.
> y

S36.  What is its name?
Please type it in.
> ships

S37.  Is SHIPS the secondary target class?
Please answer yes or no.
> n

S38.  Do you think that a class which contains some (but probably
not all) of the members of SHIPS is the target class or a
relevant class?
Please answer yes or no.
> y

S39.  The subclasses of SHIPS that are defined are:
    1.  OIL_TANKERS
    2.  MERCHANT_SHIPS
    3.  BANNED_SHIPS (ships banned from U.S. coastal waters)
    4.  SAFE_SHIPS (ships which are considered good risks)
    5.  none of the above
Which one of these is relevant?
Please enter the appropriate number.
> 3
```

S40. Is BANNED_SHIPS the secondary target class?
Please answer yes or no.
> y
Now that you have identified BANNED_SHIPS as a secondary target class, let's go back to where we were before.

S41. Is BANNED_SHIPS the class which you want the value of attribute Tanker to be in?
Please answer yes or no.
> y

S42. You have specified a constraint on the members of INSPECTIONS, by defining the class [INSPECTIONS]_WHERE_[TANKER_IS_IN_[BANNED_SHIPS]]. In a short while, you can further constrain this class if you want, i.e., by defining a class which has some of its members, or a class which contains all of its members plus others. The only issue now is to make sure that [INSPECTIONS]_WHERE_[TANKER_IS_IN_[BANNED_SHIPS]] is *relevant. Is it?*
Please answer yes or no.
> y

S43. Is [INSPECTIONS]_WHERE_[TANKER_IS_IN_[BANNED_SHIPS]] the target class?
Please answer yes or no.
> sure
Pardon?

S44. Is [INSPECTIONS]_WHERE_[TANKER_IS_IN_[BANNED_SHIPS]] the target class?
Please answer yes or no.
> y

Now that the target class has been determined, the next step is to describe the target attributes.
First, you can select aspects (attributes) of each member of [INSPECTIONS]_WHERE_[TANKER_IS_IN_[BANNED_SHIPS]] in which you are interested.

S45. Are you interested in examining attribute(s) of each member

of [INSPECTIONS]_WHERE_[TANKER_IS_IN_[BANNED_SHIPS]]?
Please answer yes or no.
> y

S46. The following member attributes of
[INSPECTIONS]_WHERE_[TANKER_IS_IN_[BANNED_SHIPS]] are
defined:

1. Tanker (the tanker inspected)
2. Date
3. Order_for_tanker (the ordering of the inspections for a tanker with the most recent inspection having value 1)
4. none of the above

Which one of these is relevant? If several are of interest,
please choose one now, and this request will repeat.
Please enter the appropriate number.
> 1

S47. Attribute Tanker of class
[INSPECTIONS]_WHERE_[TANKER_IS_IN_[BANNED_SHIPS]] cannot be
directly printed, because it is not in the form of a number
or a string of characters. Values of Tanker are members of
the class OIL_TANKERS. Thus, you may be able to print the
value of an attribute of the value(s) of Tanker of class
[INSPECTIONS]_WHERE_[TANKER_IS_IN_[BANNED_SHIPS]]. Would
you like to try to do this?
Please answer yes or no.
> y

S48. Values of Tanker are members of the class OIL_TANKERS. The
attributes of members of OIL_TANKERS are as follows:

1. Hull_type (specification of single or double hull)
2. Is_tanker_banned?
3. Inspections
4. Number_of_times_inspected
5. Last_inspection
6. Last_two_inspections
7. Date_last_examined
8. Name
9. Hull_number
10. Type (the kind of ship, e.g., merchant or fishing)
11. Country_of_registry
12. Name_of_home_port

```
    13. Cargo_types (the type(s) of cargo the ship can carry)
    14. Captain (the current captain of the ship)
    15. Engines
    16. Oil-burning_engines
    17. none of the above
Which one of these would you like to examine? If more than
one is relevant, please pick one now, and this request will
repeat.
Please enter the appropriate number.
> 8

S49. Having focused on attribute Tanker of
[INSPECTIONS]_WHERE_[TANKER_IS_IN_[BANNED_SHIPS]], you can
choose any number of attributes of members of OIL_TANKERS.
You have already selected at least one; are there any
others?
Please answer yes or no.
> y

S50. Values of Tanker are members of the class OIL_TANKERS. The
attributes of members of OIL_TANKERS are as follows:
    1. Hull_type (specification of single or double hull)
    2. Is_tanker_banned?
    3. Inspections
    4. Number_of_times_inspected
    5. Last_inspection
    6. Last_two_inspections
    7. Date_last_examined
    8. Name
    9. Hull_number
    10. Type (the kind of ship, e.g., merchant or fishing)
    11. Country_of_registry
    12. Name_of_hcme_port
    13. Cargo_types (the type(s) of cargo the ship can carry)
    14. Captain (the current captain of the ship)
    15. Engines
    16. Oil-burning_engines
    17. none of the above
Which one of these would you like to examine? If more than
one is relevant, please pick one now, and this request will
repeat.
Please enter the appropriate number.
```

```
> 13

S51. Having focused on attribute Tanker of
[INSPECTIONS]_WHERE_[TANKER_IS_IN_[BANNED_SHIPS]], you can
choose any number of attributes of members of OIL_TANKERS.
You have already selected at least one; are there any
others?
Please answer yes or no.
> n

S52. Are you interested in any other member attributes of
[INSPECTIONS]_WHERE_[TANKER_IS_IN_[BANNED_SHIPS]]?
Please answer yes or no.
> n
The target member attributes are:
    1. Name of Tanker
    2. (all values of) Cargo_types of Tanker

S53. Are you happy with this?
Please answer yes or no.
> y
Now, you can select aspects (attributes) of the class
[INSPECTIONS]_WHERE_[TANKER_IS_IN_[BANNED_SHIPS]] as a
whole, and those which describe each member of
[INSPECTIONS]_WHERE_[TANKER_IS_IN_[BANNED_SHIPS]] and which
have the same value for each member.

S54. Are you interested in examining attribute(s) of the class
[INSPECTIONS]_WHERE_[TANKER_IS_IN_[BANNED_SHIPS]] as a
whole or which have the same value for each member of
[INSPECTIONS]_WHERE_[TANKER_IS_IN_[BANNED_SHIPS]]?
Please answer yes or no.
> ?
An attribute associated with a class as a whole is one that
describes aspects of the target class taken as a unit, or
which has the same value for each member of the target
class. The former is specifically called a class
attribute; for example, a class attribute may contain as
its value the number of members in that class. The latter
is called a class-determined member attribute; for
example, the class waiters (a subclass of people, say)
might have the class-determined attribute that specifies
```

the maximum salary of each and every waiter.

S55. Are you interested in examining attribute(s) of the class [INSPECTIONS]_WHERE_[TANKER_IS_IN_[BANNED_SHIPS]] as a whole or which have the same value for each member of [INSPECTIONS]_WHERE_[TANKER_IS_IN_[BANNED_SHIPS]]?
Please answer yes or no.
> n

The transaction you have formulated is:

```
[INSPECTIONS]_WHERE_[TANKER_IS_IN_[BANNED_SHIPS]] <-
    restrict INSPECTIONS where Tanker is in [BANNED_SHIPS]

print member attributes of class
    [INSPECTIONS]_WHERE_[TANKER_IS_IN_[BANNED_SHIPS]]
    Tanker.Name,
    Tanker.Cargo_types
```

Appendix I-d. An Except from a Variation on the Dialogue in Appendix I-c

```
S39. The subclasses of SHIPS that are defined are:
    1. OIL_TANKERS
    2. MERCHANT_SHIPS
    3. BANNED_SHIPS (ships banned from U.S. coastal waters)
    4. SAFE_SHIPS (ships which are considered good risks)
    5. none of the above
Which one of these is relevant?
Please enter the appropriate number.
> 5

S40. Are you sure that you are interested in a class that
contains some of the members of SHIPS?
Please answer yes or no.
> y

S41. The class in which you are interested, that contains some
of the members of SHIPS is apparently not defined in the
data base. Would you like to try to define this subclass?
Please answer yes or no.
> y

S42. You can define a subclass by specifying another class in
the data base, so that the target class or a relevant class
contains precisely the members of SHIPS that are not in
this other class. Do you think this is possible?
Please answer yes or no.
> n

S43. You can describe a new class which contains some, but not
all, of the members of SHIPS by specifying some
constraint(s) on the value of the attributes of members of
SHIPS. Does this sound reasonable?
Please answer yes or no.
> n

S44. You can describe a new class as containing precisely all
the members of SHIPS that are currently values of some
attribute of some other class in the data base. Do you
want to try to do this?
```

```
Please answer yes or no.
> y

S45. Since you have decided to describe a new class as the one
containing precisely all the members of SHIPS that are
currently values of some attribute of some other class in
the data base, you must identify that other class and
attribute. Below is a list of the possibilities.
    1. INSPECTIONS: with attribute Tanker (of members)
    2. COUNTRIES: with attribute Ships_registered_here (of
       members)
    3. INCIDENTS: with attribute Involved_ship (of members)
    4. ASSIGNMENTS: with attribute Ship (of members)
    5. OIL_SPILLS: with attribute Involved_ship (of members)
    6. MOST_RECENT_INSPECTIONS: with attribute Tanker (of members)
    7. SHIP_TYPES: with attribute Instances (of members)
    8. CONVOYS: with attribute Oil_tanker_constituents (of
       members)
    9. CONVOYS: with attribute Constituents (of members)
    10. none of the above
Which of these is correct?
Please enter the appropriate number.
> 5

S46. Is
[SHIPS]_WHERE_[IS_A_VALUE_OF_INVOLVED_SHIP_OF_OIL_SPILLS]
the secondary target class?
Please answer yes or no.
> y
Now that you have identified
[SHIPS]_WHERE_[IS_A_VALUE_OF_INVOLVED_SHIP_OF_OIL_SPILLS]
as a secondary target class, let's go back to where we were
before.

S47. Is
[SHIPS]_WHERE_[IS_A_VALUE_OF_INVOLVED_SHIP_OF_OIL_SPILLS]
the class which you want the value of attribute Tanker to
be in?
Please answer yes or no.
> y

S48. You have specified a constraint on the members of
```

```
INSPECTIONS, by defining the class
[INSPECTIONS]_WHERE_[TANKER_IS_IN_[[SHIPS]_WHERE_
[IS_A_VALUE_OF_INVOLVED_SHIP_OF_OIL_SPILLS]]].
In a short while, you can further constrain this class if
you want, i.e., by defining a class which has some of its
members, or a class which contains all of its members plus
others.  The only issue now is to make sure that
[INSPECTIONS]_WHERE_[TANKER_IS_IN_[[SHIPS]_WHERE_
[IS_A_VALUE_OF_INVOLVED_SHIP_OF_OIL_SPILLS]]]
is relevant.  Is it?
Please answer yes or no.
> y
```

Appendix 1-e. An Except from a Variation on the Dialogue in Appendix 1-e

S33. Perhaps you want to require the value of Tanker to be present in another class. Do you?
Please answer yes or no.
> n

S34. Do you want to require the value of Tanker to be NOT present in another class?
Please answer yes or no.
> n

S35. Would you like to try to state this kind of constraint on an attribute of the value of Tanker?
Please answer yes or no.
> y

S36. Values of Tanker of INSPECTIONS are members of the class OIL_TANKERS. The attributes of members of OIL_TANKERS are as follows:

1. Hull_type (specification of single or double hull)
2. Is_tanker_banned?
3. Inspections
4. Number_of_times_inspected
5. Last_inspection
6. Last_two_inspections
7. Date_last_examined
8. Name
9. Hull_number
10. Type (the kind of ship, e.g., merchant or fishing)
11. Country_of_registry
12. Name_of_home_port
13. Cargo_types (the type(s) of cargo the ship can carry)
14. Captain (the current captain of the ship)
15. Engines
16. Oil-burning_engines
17. none of the above

Which of these is relevant?
Please enter the appropriate number.
> 14

```
S37. Perhaps you want to require the value of Captain to be
present in another class. Do you?
Please answer yes or no.
> y
In that case, the next step is to specify the class in the
data base that the value of Captain must be in.
In order to do this, you will have to identify this other
class; let us call this the SECONDARY target class. As
you will see, we will go through the same process of
finding a target class that we have been going through up
to now, except when we find this SECONDARY target class, we
will come back here and pick up where we left off.

S38. Can you name a class defined in the data base, so that the
class is the target class?
Please answer yes or no.
> n

S39. Do you know the name of a relevant class which is defined
in the data base?
Please answer yes or no.
> y

S40. What is its name?
Please type it in.
> officer

S41. Is OFFICERS the secondary target class?
Please answer yes or no.
> n

S42. Do you think that a class which contains some (but probably
not all) of the members of OFFICERS is the target class or
a relevant class?
Please answer yes or no.
> y

S43. The subclasses of OFFICERS that are defined are:
    1. DANGEROUS_CAPTAINS (captains who have been involved in an
       accident)
    2. none of the above
Which one of these is relevant?
```

```
Please enter the appropriate number.
> 2

S44. Are you sure that you are interested in a class that
contains some of the members of OFFICERS?
Please answer yes or no.
> y

S45. The class in which you are interested, that contains some
of the members of OFFICERS is apparently not defined in the
data base. Would you like to try to define this subclass?
Please answer yes or no.
> y

S46. You can define a subclass by specifying another class in
the data base, so that the target class or a relevant class
contains precisely the members of OFFICERS that are not in
this other class. Do you think this is possible?
Please answer yes or no.
> n

S47. You can describe a new class which contains some, but not
all, of the members of OFFICERS by specifying some
constraint(s) on the value of the attributes of members of
OFFICERS. Does this sound reasonable?
Please answer yes or no.
> y

S48. The primary goal now is to choose an attribute of members
of OFFICERS; then you can state a constraint on values of
that attribute. The following member attributes of
OFFICERS are defined:
    1. Name
    2. Country_of_license
    3. Date_commissioned
    4. Commander (the officer in direct command of this officer)
    5. Superiors
    6. Subordinates
    7. none of the above
You need to pick one of these attributes, so that you can
also specify some constraint its value must have in order
for a member of OFFICERS to be a member of the class being
```

defined. If more than one attribute is involved, you should pick the most important one; then later, if you want, you can impose additional constraints when this request repeats.
Please enter the appropriate number.
> 3
First, let's see if you want to compare Date_commissioned with some value you can type in.

S49. Do you wish to require the value of Date_commissioned to be equal to, not equal to, less than, less than or equal to, greater than, or greater than or equal to some value you want to type in?
Please answer yes or no.
> n
Let's see if you want to constrain the value of Date_commissioned to be in (or not in) some other class.

S50. Perhaps you want to require the value of Date_commissioned to be present in another class. Do you?
Please answer yes or no.
> n

S51. Do you want to require the value of Date_commissioned to be NOT present in another class?
Please answer yes or no.
> n
Let's see if you want to compare the value of Date_commissioned to the value of some other attribute of OFFICERS.

S52. Are you interested in comparing the value of Date_commissioned with the value of some other attribute of OFFICERS?
Please answer yes or no.
> y

S53. Now you need to to choose an attribute of the members of OFFICERS that attribute Date_commissioned is to be compared to. The following member attributes are available:
1. Name
2. none of the above

Which is the right one?
Please enter the appropriate number.
> 2

S54. Rather than comparing attribute Date_commissioned of OFFICERS to the value of another attribute of members of OFFICERS, perhaps you are interested in comparing it to a value DERIVED FROM another attribute of members of OFFICERS. Is this the case?
Please answer yes or no.
> y
In order to do so, you first need to identify this other attribute of OFFICERS, and then you can identify some attribute of it.

S55. Now you need to to choose an attribute of the members of OFFICERS that attribute Date_commissioned is to be compared to. The following member attributes are available:

1. Name
2. Country_of_license
3. Date_commissioned
4. Commander (the officer in direct command of this officer)
5. Superiors
6. Subordinates
7. none of the above

Which is the right one?
Please enter the appropriate number.
> 4

S56. It seems that you are interested in comparing Date_commissioned of OFFICERS with a value derived from Commander of OFFICERS. Is this true?
Please answer yes or no.
> y

S57. Values of Commander of OFFICERS are members of the class OFFICERS. The attributes of members of OFFICERS are as follows:

1. Name
2. Country_of_license
3. Date_commissioned
4. Commander (the officer in direct command of this officer)

```
    5. Superiors
    6. Subordinates
    7. none of the above
Which of these is relevant?
Please enter the appropriate number.
> 3

S58. Are you interested in comparing Date_commissioned and
Commander.Date_commissioned of OFFICERS?
Please answer yes or no.
> y

S59. Perhaps you want the value of Date_commissioned to be equal
to, not equal to, less than, etc., the value of
Commander.Date_commissioned. (Less than and such for
character strings means according to dictionary ordering).
Is this the case?
Please answer yes or no.
> y

S60. Several kinds of comparisons are possible:
    1. equal to
    2. not equal to
    3. greater than
    4. greater than or equal to
    5. less than
    6. less than or equal to
    7. none of the above
Which one is appropriate?
Please enter the appropriate number.
> 6

S61. You have specified a constraint on the members of OFFICERS,
by defining the class [OFFICERS]_WHERE_
[DATE_COMMISSIONED_<=_COMMANDER.DATE_COMMISSIONED].
In a short while, you can further constrain this class if
you want, i.e., by defining a class which has some of its
members, or a class which contains all of its members plus
others. The only issue now is to make sure that
[OFFICERS]_WHERE_
[DATE_COMMISSIONED_<=_COMMANDER.DATE_COMMISSIONED]
is relevant. Is it?
```

```
Please answer yes or no.
> y

S62. Is [OFFICERS]_WHERE_
[DATE_COMMISSIONED_<=_COMMANDER.DATE_COMMISSIONED]
the secondary target class?
Please answer yes or no.
> y
Now that you have identified [OFFICERS]_WHERE_
[DATE_COMMISSIONED_<=_COMMANDER.DATE_COMMISSIONED]
as a secondary target class, let's go back to where we were
before.

S63. Is [OFFICERS]_WHERE_
[DATE_COMMISSIONED_<=_COMMANDER.DATE_COMMISSIONED]
the class which you want the value of attribute Captain to
be in?
Please answer yes or no.
> y

S64. You have specified a constraint on the members of
INSPECTIONS, by defining the class
[INSPECTIONS]_WHERE_[TANKER.CAPTAIN_IS_IN_[[OFFICERS]_WHERE_
[DATE_COMMISSIONED_<=_COMMANDER.DATE_COMMISSIONED]]].
In a short while, you can further constrain this class if
you want, i.e., by defining a class which has some of its
members, or a class which contains all of its members plus
others. The only issue now is to make sure that
[INSPECTIONS]_WHERE_[TANKER.CAPTAIN_IS_IN_[[OFFICERS]_WHERE_
[DATE_COMMISSIONED_<=_COMMANDER.DATE_COMMISSIONED]]]
is relevant. Is it?
Please answer yes or no.
> y
```

Appendix 2. An Example CLU Procedure

```
find_from_base_classes = proc (tyi: file, tyo: file, s: schema,
    naive: bool, qn: int) returns (class, int)

    as = array [string]
    ac = array [class]

    ans: string
    c: class
    cn: string
    cnd: string
    i: int

    classes: ac := ac$new ()
    options: as := as$new ()
    for c in schema$base_classes (s) do
        cn := c.name
        if cn = "STRINGS" cor cn = "NUMBERS" then
            continue
            end
        if c.description ~= "" then
            cnd := cn || " (" || c.description || ")"
        else cnd := cn
            end
        as$addh (options, cnd)
        ac$addh (classes, c)
        end
    while true do
        qn, ans := ask_multiple_choice (tyi, tyo, qn,
            "Although you don't know which one, perhaps one of the" ||
            " classes defined in the data base is a relevant class," ||
            " or even the target class.  The following basic classes are" ||
            " defined:", options, "Which one of these is most relevant?")
        if ans = "?" then
            help_type_multiple_choice (tyo)
        else
            i := int$parse (ans)
            if i = ac$size (classes) + 1 then
                say (tyo, "You have to start with some class.  Let's try again.")
            elseif i >= 1 cand i <= ac$size (classes) then
```

```
                c := classes [i]
                return (c, qn)
            else
                say (tyo, "Sorry, what did you say?")
                end
            end
        end
end find_from_base_classes
```

OFFICIAL DISTRIBUTION LIST

Defense Documentation Center
Cameron Station
Alexandria, VA 22314
12 copies

Office of Naval Research
Information Systems Program
Code 437
Arlington, VA 22217
2 copies

Office of Naval Research
Branch Office/Boston
495 Summer Street
Boston, MA 02210
1 copy

Office of Naval Research
Branch Office/Chicago
536 South Clark Street
Chicago, IL 60605
1 copy

Office of Naval Research
Branch Office/Pasadena
1030 East Green Street
Pasadena, CA 91106
1 copy

New York Area Office
715 Broadway - 5th floor
New York, N. Y. 10003
1 copy

Naval Research Laboratory
Technical Information Division
Code 2627
Washington, D. C. 20375
6 copies

Assistant Chief for Technology
Office of Naval Research
Code 200
Arlington, VA 22217
1 copy

Office of Naval Research
Code 455
Arlington, VA 22217
1 copy

Dr. A. L. Slafkosky
Scientific Advisor
Commandant of the Marine Corps
(Code RD-1)
Washington, D. C. 20380
1 copy

Office of Naval Research
Code 458
Arlington, VA 22217
1 copy

Naval Electronics Lab Center
Advanced Software Technology
Division - Code 5200
San Diego, CA 92152
1 copy

Mr. E. H. Gleissner
Naval Ship Research & Development Center
Computation & Math Department
Bethesda, MD 20084
1 copy

Captain Grace M. Hopper
NAICOM/MIS Planning Branch
(OP-916D)
Office of Chief of Naval Operations
Washington, D. C. 20350
1 copy

Captain Richard L. Martin, USN
Commanding Officer
USS Francis Marion (LPA-249)
FPO New York, N. Y. 09501
1 copy